Modellierung und Generierung kombinierbarer Benutzungsschnittstellenvarianten und deren gemeinschaftliche Nutzung in Dienst-Ökosystemen

---

Dipl.-Inform. Michael Hitz

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur
(Dr.-Ing.)
in der Wissenschaftsdisziplin
"Komplexe Multimediale Anwendungsarchitekturen"

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
Institut für Informatik und Computational Science, LE Informatik
der Universität Potsdam

Ort und Tag der Disputation: Potsdam, den 1. März 2021

Hauptbetreuerin:
Prof. Dr.-Ing. habil. Ulrike Lucke
Betreuer:
Prof. Dr. Thomas Kessel
Gutachter*innen:
Prof. Dr.-Ing. habil. Ulrike Lucke
Prof. Dr. Thomas Kessel
Prof. Dr.-Ing. Jürgen Ziegler

Online veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
https://doi.org/10.25932/publishup-50022
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-500224

Zusammenfassung

Digitalisierung ermöglicht es uns, mit Partnern (z.B. Unternehmen, Institutionen) in einer IT-unterstützten Umgebung zu interagieren und Tätigkeiten auszuführen, die vormals manuell erledigt wurden. Ein Ziel der Digitalisierung ist dabei, Dienstleistungen unterschiedlicher fachlicher Domänen zu Prozessen zu kombinieren und vielen Nutzergruppen bedarfsgerecht zugänglich zu machen. Hierzu stellen Anbieter technische Dienste bereit, die in unterschiedliche Anwendungen integriert werden können.

Die Digitalisierung stellt die Anwendungsentwicklung vor neue Herausforderungen. Ein Aspekt ist die bedarfsgerechte Anbindung von Nutzern an Dienste. Zur Interaktion menschlicher Nutzer mit den Diensten werden Benutzungsschnittstellen benötigt, die auf deren Bedürfnisse zugeschnitten sind. Hierzu werden Varianten für spezifische Nutzergruppen (fachliche Varianten) und variierende Umgebungen (technische Varianten) benötigt. Zunehmend müssen diese mit Diensten anderer Anbieter kombiniert werden können, um domänenübergreifend Prozesse zu Anwendungen mit einem erhöhten Mehrwert für den Endnutzer zu verknüpfen (z.B. eine Flugbuchung mit einer optionalen Reiseversicherung).

Die Vielfältigkeit der Varianten lässt die Erstellung von Benutzungsschnittstellen komplex und die Ergebnisse sehr individuell erscheinen. Daher werden die Varianten in der Praxis vorwiegend manuell erstellt. Dies führt zur parallelen Entwicklung einer Vielzahl sehr ähnlicher Anwendungen, die nur geringes Potential zur Wiederverwendung besitzen. Die Folge sind hohe Aufwände bei Erstellung und Wartung. Dadurch wird häufig auf die Unterstützung kleiner Nutzerkreise mit speziellen Anforderungen verzichtet (z.B. Menschen mit physischen Einschränkungen), sodass diese weiterhin von der Digitalisierung ausgeschlossen bleiben.

Die Arbeit stellt eine konsistente Lösung für diese neuen Herausforderungen mit den Mitteln der modellgetriebenen Entwicklung vor. Sie präsentiert einen Ansatz zur Modellierung von Benutzungsschnittstellen, Varianten und Kompositionen und deren automatischer Generierung für digitale Dienste in einem verteilten Umfeld. Die Arbeit schafft eine Lösung zur Wiederverwendung und gemeinschaftlichen Nutzung von Benutzungsschnittstellen über Anbietergrenzen hinweg. Sie führt zu einer Infrastruktur, in der eine Vielzahl von Anbietern ihre Expertise in gemeinschaftliche Anwendungen einbringen können.

Die Beiträge bestehen im Einzelnen in Konzepten und Metamodellen zur Modellierung von Benutzungsschnittstellen, Varianten und Kompositionen sowie einem Verfahren zu deren vollständig automatisierten Transformation in funktionale Benutzungsschnittstellen. Zur Umsetzung der gemeinschaftlichen Nutzbarkeit werden diese ergänzt um eine universelle Repräsentation der Modelle, einer Methodik zur Anbindung unterschiedlicher Dienst-Anbieter sowie einer Architektur zur verteilten Nutzung der Artefakte und Verfahren in einer dienstorientierten Umgebung.

Der Ansatz bietet die Chance, unterschiedlichste Menschen bedarfsgerecht an der Digitalisierung teilhaben zu lassen. Damit setzt die Arbeit Impulse für zukünftige Methoden zur Anwendungserstellung in einem zunehmend vielfältigen Umfeld.

Abstract

Digitalization enables us to interact with partners (e.g., companies, institutions) and perform operations in an IT-supported, digital environment. A major objective of the digitalization efforts is to combine and integrate services of different business domains in order to make them accessible to a wide variety of users. To achieve that objective, service suppliers provide technical services that can be integrated into applications.

This brings new challenges for the development of applications. An important aspect is the needs-oriented accessibility of services for users. To interact with technical services, human users need user interfaces (UIs) tailored to suit their needs. This requires a variety of UI variants focusing on the different requirements of special user groups (user specific variants) and tailored for multiple technical environments (technical variants). In addition, user interfaces increasingly need to combine services from different suppliers and business domains to build applications with added value for the customer (e.g., a flight booking in combination with a travel insurance product).

The diversity of the variants lets the development of user interfaces seems to be a complex and very individual task. Therefore, in daily practice variants are mostly built manually. This leads to concurrent developments of very similar, yet hardly reusable applications. This results in increasing expenses for application building and maintenance. One consequence is that service providers shy away from the effort to build variants for small user groups with special needs (e.g., visually impaired persons), excluding them from digital offerings and leaving them behind.

This thesis offers a consistent solution for these new challenges by applying model driven development techniques to the problem. It presents an approach for modeling dialog-based user interfaces, variants and compositions including their completely automatic generation as UIs for digital services in a distributed environment. The thesis provides a solution for reuse and sharing of user interfaces across different domains and suppliers. It leads to an infrastructure where a variety of suppliers can share their expertise to build applications.

The contributions of this work are concepts and metamodels for modeling user interfaces, variants and compositions along with a process for a completely automated transformation into fully functional UIs. To achieve reusability and shareability the solutions are complemented by a universal representation of the models, a method to reuse them with different service suppliers and a distributed architecture for sharing the artifacts and functional components of the approach in a service oriented environment.

The approach offers the opportunity to a variety of people to participate in digitalization. It provides impetus for future application development methods within an increasingly diverse environment.

Danksagung

"Niemand kann sich seine Familie aussuchen!" sagt man. Ich hatte das, Glück, dass mir dies für meine Promotionsfamilie dennoch möglich war - und bin dankbar, dass ich Menschen gefunden habe, die sich für mich und mein Vorhaben so außergewöhnlich eingesetzt haben.

Mein besonderer Dank gilt Frau Prof. Dr.-Ing. Lucke, die mich als Doktorand angenommen und meiner Arbeit so viel Vertrauen entgegengebracht hat. Sie gab mir das Umfeld, die Unterstützung und persönliche Sicherheit, die meine Arbeit ins Ziel gebracht hat. Auch Prof. Dr. Kessel gilt mein spezieller Dank. Er begleitete mich von Anfang an und war als Betreuer in jeder Phase - jedem Hoch und Tief - auf fachlicher und menschlicher Ebene für mich da.

Prof. Dr. Grunske möchte ich für die wertvollen Hinweise gegen Ende der Arbeit danken. Vielen Dank zudem an Prof. Dr.-Ing. Jürgen Ziegler für die sehr kurz entschlossene Bereitschaft, diese Arbeit als Gutachter zu unterstützen.

Der Dualen Hochschule Baden-Württemberg (DHBW Stuttgart), insbesondere Prof. Dr. Müllerschön, möchte ich dafür danken, dass mir dort als wissenschaftlicher Mitarbeiter die Umgebung zur Durchführung der Arbeit gegeben wurde. Prof. Dr.-Ing. Pfisterer danke ich für die ersten Schritte und Gespräche zum Aufbau der Arbeit und die fachlichen Impulse zu Ontologien.

Prof. Dr. Schwenkreis danke ich dafür, dass er mich als Zimmerkollegen aufgenommen und mir in vielen Gesprächen zu Ideen verhalf, wenn es mal klemmte. Und natürlich auch dafür, dass er maßgeblich dazu beitrug, einen *sicheren Hafen* für die Arbeit zu finden.

Mein Dank gilt auch Mirjana Radonjic-Simic, die als *Doktorschwester* fachlich und menschlich eine Ergänzung war und mit mir gemeinsam ein großes Stück des Weges bestritt.

Auch danke ich meinen Kollegen bei der Allianz Deutschland AG, welche mir mit vielen fachlichen Gesprächen und wertvollen Diskussionen halfen, Muster in unserem Tun zu finden und mein Wissen zu erweitern. Auch möchte ich meinen Vorgesetzten danken, insbesondere Thomas Stagel, die mir den Raum zur Durchführung der Arbeit geschaffen haben.

Nicht zuletzt ist es die *andere Familie*, die einem den Raum für ein solches Vorhaben schafft. Ich danke meinen Eltern, dass sie mir in der Vergangenheit die Freiheit gaben, meinen Weg zu finden - und mich auch danach in allen Vorhaben bestärkt haben. Ich danke der Familie meiner Partnerin für ihre Unterstützung und zeitweise dringend notwendige Ablenkung.

Aber ohne den Rückhalt meiner Partnerin Stefanie Friedrich wäre die Arbeit nicht möglich gewesen. Vielen Dank für Deine Liebe, Hilfe, Geduld und Ausdauer!

Inhalt

1. Einleitung

Die fortschreitende Digitalisierung der vergangenen Jahre führt dazu, dass Unternehmen vermehrt Geschäftsprozesse automatisieren und diese als Dienste (z.B. in Form technischer Web Services) Endnutzern bereitstellen. Beispiele solcher Dienste begegnen uns täglich, z.B. die Buchung eines Fluges, das Rufen eines Taxis, das Bestellen von Waren und Dienstleistungen oder der Antrag für ein Versicherungsprodukt.

Die Bereitstellung von Geschäftsprozessen bietet Unternehmen einen mehrfachen Nutzen. Einerseits können Kunden direkt mit den Systemen des Unternehmens interagieren und so Geschäftsprozesse selbständig durchführen. Dies führt zur Einsparung von Personalkosten und einer Beschleunigung der Durchführung von Prozessen. Andererseits eröffnet sie Dritten die Möglichkeit zur Entwicklung neuer Geschäftsmodelle. Dienstleistungen von Unternehmen unterschiedlicher fachlicher Domänen können kombiniert und in eigene Anwendungen integriert werden (z.B. die Flugbuchung mit kombinierter Reiseversicherung). Anbieter wiederverwendbarer Dienste existieren bereits in vielen Bereichen: z.B. Transport und Verkehr[1] (*Uber.com*, *bahn.de*, *flightstats.com*), Finanzdienstleistungen[2] (*PayPal.de*, *VISA.com*) oder Versicherungen (Reise-, Kfz- oder Transportversicherungen).

Um das Zusammenspiel dieser Dienste zu gewährleisten und eine Austauschbarkeit zu erreichen, entstanden in Forschung und Wirtschaft in den vergangenen Jahren Standardisierungs-Initiativen, die eine gemeinsame Konzeptionalisierung der Daten und Prozesse für bestimmte Domänen zum Ziel haben (z.B. die BiPRO-Initiative im Versicherungsbereich[3]). Dies ermöglichte die Entstehung einer Vielzahl funktional gleichartiger Dienste unterschiedlicher Anbieter, die auf gemeinsamen Prozess- und Datenbeschreibungen gründen [146]. Sie bilden Ökosysteme domänenspezifischer Dienste (sog. **Dienst-Ökosysteme** [65]), die miteinander interagieren und zu neuen Anwendungen kombinierbar sind.

1.1 Benutzungsschnittstellen in Dienst-Ökosystemen

Zur Interaktion eines Nutzers mit diesen Diensten werden **Benutzungsschnittstellen** (engl. *User Interface*, Abk. *UI*) benötigt, welche die zur Durchführung von Geschäftsprozessen notwendigen Daten erfassen. Deren Erstellung für Dienste in einem Ökosystem stellt aufgrund der Vielzahl möglicher Nutzungsszenarien eine besondere Herausforderung dar. Wenngleich die Dienste auf einer standardisierten Konzeptionalisierung beruhen, so variieren die Benutzungsschnittstellen abhängig von der Nutzung. Herausforderungen sind insbesondere:

- Unterstützung unterschiedlicher Nutzergruppen
- Bereitstellung für verschiedene Interaktionsformen und Technologien
- Kombination von Diensten in Benutzungsschnittstellen
- Anbindung verschiedener Dienst-Anbieter

Benutzungsschnittstellen müssen in Varianten für unterschiedliche Nutzergruppen und multiple Plattformen bereitgestellt werden, die sich inhaltlich und technologisch unterscheiden

[1] http://developer.uber.com, https://developer.deutschebahn.com, https://developer.flightstats.com/

[2] https://developer.paypal.com, https://developer.visa.com/

[3] http://bipro.net

(z.B. für Endkunden oder Innendienst als Desktop-, browserbasierte oder sprachgesteuerte Anwendungen). Zudem werden meist mehrere Dienste zur Umsetzung eines Szenarios in einer Benutzungsschnittstelle kombiniert. Um die Vorteile eines Dienst-Ökosystems nutzen zu können, müssen sie darüber hinaus mit Dienstimplementierungen beliebiger Anbieter zusammenarbeiten können.

Ein konkretes Beispiel für eine Anwendung, die für vielfältige Nutzergruppen und Plattformen bereitgestellt werden muss, ist die Antragstrecke für eine Haftpflichtversicherung. Diese existiert sowohl in einer umfangreichen Variante, die von Vermittlern zur Angebotserstellung genutzt, als auch in einer vereinfachten Version, die von Endkunden im Internet als *self service* aufgerufen werden kann. Ein Beispiel für eine kombinierte Anwendung aus Bausteinen unterschiedlicher Domänen könnte eine Reisebuchung sein, in deren Verlauf eine Reiseversicherung abgeschlossen werden kann. Die Beispiele werden in den Abschnitten 2.2.1, 2.3.1 und 2.4.1 detailliert dargestellt. Sie dienen im weiteren Verlauf der Arbeit wiederkehrend zur Illustration.

1.2 Problemstellung

Die genannten Herausforderungen und die daraus resultierende Komplexität führen dazu, dass Benutzungsschnittstellen vorwiegend manuell erstellt werden. Sie werden für spezifische Nutzergruppen, Plattformen und Dienst-Anbieter gebaut und eignen sich dadurch meist nicht zur Wiederverwendung in verschiedenen Szenarien oder als Komponenten. Dies führt zu einer **Mehrfachentwicklung gleichartiger Anwendungen**, die in *Silos* [11] nebeneinander existieren. Dies vervielfacht den Aufwand bei der Erstellung und erhöht die Fehleranfälligkeit bei Entwicklung und Wartung aufgrund der Redundanzen [172].

Eine Lösung für dieses Problem verspricht die **modellgetriebene Entwicklung**. Der Grundgedanke besteht darin, abstrakte Modelle der Benutzungsschnittstellen zu erstellen und daraus Varianten automatisiert zu generieren. Um den genannten Herausforderungen mit der modellgetriebenen Entwicklung zu begegnen, bedarf es eines Ansatzes, der folgende Eigenschaften aufweist:

- Erzeugung von Benutzungsschnittstellen, die funktional manuell erstellten entsprechen
- Modellierbarkeit von Varianten für verschiedene Nutzergruppen
- Generierbarkeit von Varianten für multiple Plattformen und Technologien

Die Wiederverwendbarkeit von Modellen trägt wesentlich zur Vermeidung einer Mehrfachentwicklung bei. Insbesondere in einem Dienst-Ökosystem erfordert dies die Kombinierbarkeit bestehender Benutzungsschnittstellen, die mit beliebigen Dienst-Anbietern interagieren können. Weitere notwendige Eigenschaften sind daher:

- Komposition und Anpassung bestehender Modelle in neuen Umgebungen
- Gemeinschaftliche Nutzung bestehender Modelle anderer Anbieter
- Verwendbarkeit von Benutzungsschnittstellen mit Diensten beliebiger Anbieter

Zur modellgetriebenen Entwicklung von Benutzungsschnittstellen werden in der Literatur verschiedene Ansätze vorgeschlagen. Diese bieten jedoch **nur partielle Lösungen für die**

genannten Problemstellungen. Sie fokussieren sich vorwiegend auf die Modellierung UI-spezifischer Ergebnistypen (z.B. Dialogseiten, konkrete Dialogabläufe) und erzeugen daraus Benutzungsschnittstellen, die sich meist auf bestimmte Technologiekategorien beschränken (z.B. grafische Oberflächen für Web-Anwendungen). Sie beinhalten meist keine Mechanismen zur redundanzfreien Modellierung inhaltlicher Varianten oder Kompositionen. Die zugrundeliegenden Modelle eignen sich daher nur bedingt zur Erstellung wiederverwendbarer Varianten für verschiedene Nutzergruppen und Modalitäten [43]. Zudem erfordern die Ansätze die manuelle Erstellung einer z.T. großen Zahl voneinander abhängiger Artefakte[4], was die Entwicklung und Wartung erschwert. Der Aspekt der Nutzung in einem Dienst-Ökosystem ist bisher weitgehend unbeachtet. Hier existieren Lösungen für Teilprobleme (z.B. Aggregation grafischer UIs als *Mashups* [143]), die jedoch kein durchgängiges Konzept für eine gemeinschaftliche Nutzung bieten. Aufgrund der Einschränkungen und Komplexität finden die Ansätze bisher nur geringe Akzeptanz in der Praxis [140].

Nach meinem besten Wissen existieren keine Ansätze, welche die genannten Eigenschaften in ihrer Gesamtheit besitzen. Meine Erfahrungen der vergangenen Jahre als Software-Architekt bei der Allianz Deutschland AG zeigen den Mangel an Lösungen. Die Motivation für die vorliegende Arbeit besteht in der Schließung der Forschungslücken mit einem Ansatz, der in der Praxis nutzbar ist.

1.3 Ziele und Forschungsfragen

Das Ziel der Arbeit ist die Entwicklung eines durchgängigen Ansatzes zur modellgetriebenen Entwicklung dialogbasierter Benutzungsschnittstellen. Es wird ein Ansatz vorgeschlagen, der die Modellierung von Benutzungsschnittstellen, Varianten und deren Komposition ermöglicht sowie die vollständig automatisierte Erzeugung lauffähiger Benutzungsschnittstellen erlaubt. Die vorgeschlagenen Modelle sind wiederverwendbar und können an den Kontext ihrer Wiederverwendung angepasst werden. Durch die vorgeschlagene Lösung sollen folgende Verbesserungen im Vergleich zu bestehenden Ansätzen erreicht werden:

- Vereinfachung der Erstellung und Wartung von Benutzungsschnittstellen
- Geringere Komplexität und Redundanz bei der Modellierung von Varianten
- Automatische Erzeugung nicht-trivialer Benutzungsschnittstellen
- Erhöhte Wiederverwendbarkeit der Modelle durch Kombinierbarkeit in variierenden Szenarien
- Gemeinschaftliche Nutzbarkeit der Modelle in Dienst-Ökosystemen

Hierzu wird ein Ansatz zur **datenzentrierten Modellierung von Benutzungsschnittstellen** vorgeschlagen. Die Kernidee besteht darin, die von einer Anwendung verarbeiteten Daten, deren Beziehungen und deren Semantik in einem Modell zu beschreiben. Es dient als technologieneutrale Grundlage zur Beschreibung von Anwendungen. Zur Erzeugung einer konkreten Benutzungsschnittstelle wird dieses Modell automatisiert um Technologiespezifika angereichert und daraus eine funktionsfähige Anwendung für eine spezifische Nutzungsumgebung generiert.

[4] Unter *Artefakt* (engl. *artifact*) wird im Folgenden ein manuell oder maschinell erstelltes Ergebnis aus einem Arbeitsprozess der Softwareentwicklung verstanden, z.B. Konstrukte/Konzepte, Modelle, Verfahren, Architekturen (vgl. [52, 212]).

Die Verwendung eines datenzentrierten Modells vereinfacht die Erstellung und Wartung von Varianten. Varianten werden im vorgeschlagenen Ansatz über die Beschreibung der Unterschiede zu einem gemeinsamen Anwendungsmodell beschrieben. So werden Redundanzen vermieden. Auch die Wiederverwendbarkeit der Modelle vereinfacht sich durch die Datenzentrierung. Bestehende Modelle können unabhängig von der verwendeten Technologie referenziert und in ein neues, wiederum datenzentriertes Modell integriert werden. Aufgrund der Technologieneutralität der Modellierung wird eine gemeinschaftliche Nutzung ermöglicht. Die datenzentrierten Modelle können verteilt und in unterschiedlichen technologischen Umgebungen verwendet werden.

Aus der Kernidee ergeben sich die **Forschungsfragen** für die Arbeit. Tabelle 1.1 stellt die mit (FF.1) - (FF.5) bezeichneten Hauptfragen dar, die durch Detailfragen ergänzt werden. Sie umfassen die Problembereiche, die zur Umsetzung der Lösungsidee zu behandeln sind. Insbesondere sind dies Fragestellungen zur Umsetzung der datenzentrierten Modellierung, Spezifikation von Varianten und Kompositionen, zur Generierung finaler Benutzungsschnittstellen und der gemeinschaftliche Nutzung und Wiederverwendung.

	Forschungsfragen
(FF.1)	**Wie können dialogbasierte Benutzungsschnittstellen und Varianten technologieneutral beschreiben werden?**
	▪ Welche Informationen werden zur Generierung einer nicht-trivialen Benutzungsschnittstelle benötigt? ▪ Welche Informationen sind zur Herleitung inhaltlicher und technischer Varianten erforderlich? ▪ Können diese Informationen *technologieneutral* formuliert werden, sodass sie auf multiple Interaktions- und Technologiekontexte anwendbar sind?
(FF.2)	**Wie können Benutzungsschnittstellen und Varianten modelliert werden?**
	▪ Wie können die Erkenntnisse aus (FF.1) in einem *einfachen* Modell abgebildet werden? ▪ Wie können auf diesem Modell fachliche Varianten *redundanzfrei* definiert werden?
(FF.3)	**Wie können Anwendungsmodelle zu komplexeren Anwendungen kombiniert werden?**
	▪ Wie können Modelle und Varianten als Komponenten zu vollständigen Anwendungen aggregiert werden? ▪ Wie können Komponenten an ihre neue Nutzung angepasst werden?
(FF.4)	**Wie lassen sich aus einem Anwendungsmodell technische Varianten automatisch herleiten?**
	▪ Welche Schritte sind zur Herleitung von Anwendungen, Varianten und Kompositionen notwendig? ▪ Wie wird technische Variabilität für multiple Interaktions- und Technologiekontexte erreicht?
(FF.5)	**Wie können Benutzungsschnittstellen in Dienst-Ökosystemen gemeinschaftlich wiederverwendet werden?**
	▪ Was sind die Voraussetzungen für eine gemeinschaftliche Nutzung? ▪ Wie können die Modelle zur gemeinschaftlichen Nutzung bereitgestellt und verarbeitet werden? ▪ Wie können Dienste an Benutzungsschnittstellen angebunden werden?

Tabelle 1.1. Forschungsfragen der Arbeit

1.4 Beiträge und Aufbau der Arbeit

Mit der Beantwortung der Forschungsfragen ergänzt die Arbeit bisherige Forschungen um Beiträge in den Themenfeldern der technologieneutralen Modellierung von Benutzungsschnittstellen, Varianten und Kompositionen. Zudem fügt sie neue Ansätze zur gemeinschaftlichen Nutzung von Benutzungsschnittstellen und deren Verwendung in Dienst-Ökosystemen hinzu. Der Hauptbeitrag liegt in der Erarbeitung eines datenzentrierten, modellgetriebenen Ansatzes zur Generierung von Benutzungsschnittstellen und Varianten dialogbasierter Anwendungen. Einzelbeiträge sind Folgende:

1. Systematische Analyse zur Beschreibung dialogbasierter Benutzungsschnittstellen benötigter Informationen
2. Ansatz zur datenzentrierten Modellierung von Benutzungsschnittstellen
3. Methodik zur Spezifikation inhaltlicher Varianten
4. Methodik zur Modellierung von Kompositionen
5. Verfahren zur automatischen Herleitung technischer Varianten
6. Ansatz zur Modellierung gemeinschaftlich nutz- und wiederverwendbarer Benutzungsschnittstellen in Dienst-Ökosystemen
7. Architektur zu deren verteilten Nutzung

Die Neuerung des vorgeschlagenen Ansatzes besteht in der datenzentrierten Modellierung von Benutzungsschnittstellen. Diese führt zu einer vereinfachten Modellierung kombinierbarer Anwendungsvarianten. Sie ermöglicht die gemeinschaftliche Wiederverwendung und resultiert in einem neuartigen Ansatz zur Erstellung von Benutzungsschnittstellen für dialogbasierte Anwendungen, der mit bestehenden Ansätzen nicht erreicht werden kann.

Kapitel 2 beschreibt den Problembereich der Arbeit und erläutert Grundlagen und Begriffe, die zum weiteren Verständnis der Arbeit notwendig sind. In **Kapitel 3** werden daraus die Anforderungen an den Lösungsansatz abgeleitet. Diese werden dem Stand der Forschung gegenübergestellt und dienen zur Identifikation der Forschungslücke. In der Folge wird der verfolgte Lösungsansatz im Überblick sowie das Realisierungskonzept vorgestellt, die in den folgenden Kapiteln umgesetzt werden.

Kapitel 4 stellt die Methodik vor, nach welcher der Lösungsansatz der Arbeit entwickelt und die Ergebnisse validiert werden. Es wird das zur Durchführung der Arbeit angewandte Forschungs-Paradigma und Vorgehensmodell eingeführt und deren Anwendung auf die konkreten Elemente des skizzierten Lösungsansatzes dargestellt. Zudem wird hergeleitet, anhand welcher Methoden und Kriterien die abschließende Validierung und Evaluation der Ergebnisse erfolgt.

Kapitel 5 schafft die Grundlagen zur Modellierung, indem die Eigenschaften dialogbasierter Benutzungsschnittstellen sowie inhaltlicher und technischer Varianten analysiert werden. Daraus werden Anforderungen an die Modellierung abgeleitet. Das Ergebnis besteht in einer systematischen Darstellung der Informationen, die zur automatischen Erzeugung von Benutzungsschnittstellenvarianten erforderlich sind.

Kapitel 6 stellt den Ansatz zur datenzentrierten Modellierung dialogbasierter Benutzungsschnittstellen sowie inhaltlicher Varianten vor. Das Ergebnis besteht in einem Metamodell zur Beschreibung von Benutzungsschnittstellen, einer Methodik zur redundanzfreien Spezi-

fikation von Varianten, sowie einer *Domain Specific Language* (DSL), die als Notation zur Modellierung vorgeschlagen wird.

Kapitel 7 beschreibt die Methodik zur Modellierung von Kompositionen. Das Ergebnis besteht in der Erweiterung des Metamodells um ein Konzept zur Integration von Komponenten sowie der Methodik zu deren Anpassung an ein neues Umfeld.

Kapitel 8 stellt ein Verfahren zur automatischen Generierung finaler Benutzungsschnittstellen aus datenzentrierten Modellen vor. Es stellt die Schritte dar, in denen aus den Modellen der Kapitel 6 und 7 technische Benutzungsschnittstellenvarianten erzeugt werden.

In **Kapitel 9** werden die Ergebnisse auf die Erstellung von Anwendungen in Dienst-Ökosystemen übertragen und um Konzepte zur universellen Beschreibung von Anwendungsmodellen und zur generischen Dienstanbindung erweitert. Zudem wird aufgezeigt, wie darauf aufbauend eine dezentrale Infrastruktur geschaffen werden kann, die eine gemeinschaftliche Nutzung der Modelle in Dienst-Ökosystemen ermöglicht.

Kapitel 10 beschreibt die Realisierung und Evaluation der Ergebnisse. Hierzu wird im ersten Schritt die Auswahl geeigneter Nachweismethoden dargestellt und anschließend die Machbarkeit, Anwendbarkeit und Nutzbarkeit der erstellten Ergebnisse nachgewiesen.

Kapitel 11 fasst die Ergebnisse zusammen und gibt ein Ausblick auf zukünftige Arbeiten und weitere Anwendungsbereiche des vorgestellten Ansatzes.

Das Promotionsvorhaben wurde unter dem Projektnamen ***mimesis*** durchgeführt, unter dem auch Veröffentlichungen zu Teilergebnissen erfolgten. Im Verlauf der Arbeit findet sich der Name an einigen Stellen wieder - insbesondere bei der Darstellung der Ergebnistypen in Beispielen und der Umsetzung in der Evaluation.

Der Begriff *mimesis* ist der Zoologie entlehnt. Als *Mimese*[5] (griechisch: $\mu\iota\mu\eta\sigma\iota\varsigma$, mímēsis, Nachahmung) wird dort eine Form der Tarnung bezeichnet, bei der ein Lebewesen Gestalt, Farbe und Haltung eines Teil seines Lebensraumes annimmt - und so vom Umfeld nicht mehr unterschieden werden kann. Diese Eigenschaft lässt sich auf Benutzungsschnittstellen übertragen, die mit dem vorgestellten Ansatz erstellt werden. Es werden mimetische UIs modelliert, die sich in eine Umgebung einbetten lassen und dabei Form und Verhalten an ihren neuen Lebensraum anpassen.

[5] https://www.duden.de/rechtschreibung/Mimese

2. Problembereich und Grundlagen

Dieses Kapitel führt den Problembereich und Begriffe ein, die in der Arbeit wiederkehrend verwendet werden. Die Themenfelder sind:

- Dienste, Serviceorientierte Architekturen und Dienst-Ökosysteme (Abschnitt 2.1)
- Dialogbasierte Anwendungen (Abschnitt 2.2)
- Varianten dialogbasierter Anwendungen (Abschnitt 2.3)
- Komposition von Anwendungen (Abschnitt 2.4)
- *Shared UIs*: Wiederverwendung in Dienst-Ökosystemen (Abschnitt 2.5)
- Modellgetriebene Entwicklung von Benutzungsschnittstellen (Abschnitt 2.6)

Die Abschnitte beschreiben Konzepte in diesen Themenfeldern. Sie nennen Eigenschaften, die im weiteren Verlauf als Grundlage zur Formulierung der Anforderungen an einen modellgetriebenen Ansatz referenziert werden.

2.1 Dienste, Serviceorientierte Architekturen und Dienst-Ökosysteme

Der Begriff der *Dienstorientierung* (*service orientation*) stammt ursprünglich aus dem betriebswirtschaftlichen Umfeld. Der Druck, schnell auf sich verändernde Marktbedingungen reagieren zu können und die Notwendigkeit, Teile der Geschäftsprozesse an Partner auslagern zu können, erforderte eine Modularisierung der Leistungen eines Unternehmens und damit eine Anpassung der Unternehmensstruktur. Als Mittel zur Strukturierung wurde eine *serviceorientierte Unternehmensstruktur* vorgeschlagen [135]. Die zugrundeliegende Idee ist, die Geschäftsprozesse in eine Menge von *Services*[6] aufzuteilen, die zur Erbringung der Dienstleistungen und Erstellung der Produkte eines Unternehmens kombiniert werden können. Die Zerlegung in funktionale Einheiten ermöglicht deren Wiederverwendung, deren Auslagerung an Partner bzw. das Angebot eigener Dienstleistungen an Dritte [33].

Im Zuge der Digitalisierung wurden diese Ansätze auf die IT-Architekturen übertragen. So ist der Dienstgedanke die Basis z.B. für den Aufbau verteilter Systeme [55, 221] oder prozessbasierter Workflow-Management-Anwendungen [91, 111]. Zudem wurden die technischen Ansätze zur Verteilung von Programmlogik (z.B. *Remote Procedure Calls - RPC* [221]) im Sinne des Dienstgedankens zur Erstellung **digitaler Dienste** weiterentwickelt und die Nutzung durch Verwendung von Webtechnologien vereinfacht und standardisiert[7].

Die *Open Group*[8] definiert einen Dienst als eine in sich geschlossene Aktivität (*self-contained*), die ggf. aus anderen Diensten zusammengesetzt sein kann (*composition*). Um einen Dienst in verschiedenen Kontexten nutzbar zu machen, nennt die *Organization for the Advancement of Structured Information Standards (OASIS)* [155] als zentrale Eigenschaften dessen Spezifikation über eine **definierte und beschriebene Schnittstelle** und dessen

[6] Die Begriffe *Dienst* und *Service* werden synonym verwendet. Da *Service* jedoch häufig im Zusammenhang serviceorientierter IT-Architekturen mit *WebServices* und damit konkreten Technologien assoziiert wird, wird im Rahmen der Arbeit bevorzugt der Begriff *Dienst* als allgemeinere Variante verwendet.

[7] vgl. z.B. https://www.w3.org/TR/soap12/

[8] http://www.opengroup.org/soa/source-book/soa/p1.htm

kombinierte Verwendbarkeit in Zusammenhängen, die über die vom Anbieter ursprüngliche beabsichtigte Nutzung hinausgehen[9]. Das Fundament sind Schnittstellendefinitionen, welche die Semantik, Ein- und Ausgabeparameter und Voraussetzungen für die Nutzung festlegen [114, 129].

2.1.1 Serviceorientierte Architekturen zur Wiederverwendung

Aus Unternehmenssicht liegt das Ziel in der Wiederverwendung und Kombination von IT-Diensten, die in mehreren Anwendungen genutzt werden und über multiple technische Kanäle und für unterschiedliche Nutzergruppen erreichbar sind [132]. Dies erfordert eine Anwendungsarchitektur, die auf eine entsprechende Infrastruktur zu deren Nutzung zurückgreifen kann [4, 126] - eine multikanalfähige Architektur zum Aufbau dienstbasierter Unternehmensanwendungen [96, 97, 198]. In Gartner [202, 203] wurde hierfür der Begriff **Serviceorientierte Architektur (SOA)** eingeführt. In Josuttis [114] wird eine SOA als Architektur-Ansatz definiert, in welchem existierende, von unterschiedlichen Anbietern bereitgestellte Dienste in einer heterogenen System-Infrastruktur wiederverwendet werden.

Eine SOA setzt eine Infrastruktur voraus, welche das Auffinden und Ausführen von Diensten auf die entfernten Systemen ermöglicht. Abbildung 2.1 (mit Änderungen aus Josuttis [114]) stellt die Umsetzung einer SOA dar. Sie zeigt die Trennung von Dienst-Nutzer und -Anbieter sowie die erforderlichen Infrastrukturkomponenten auf konzeptioneller Ebene.

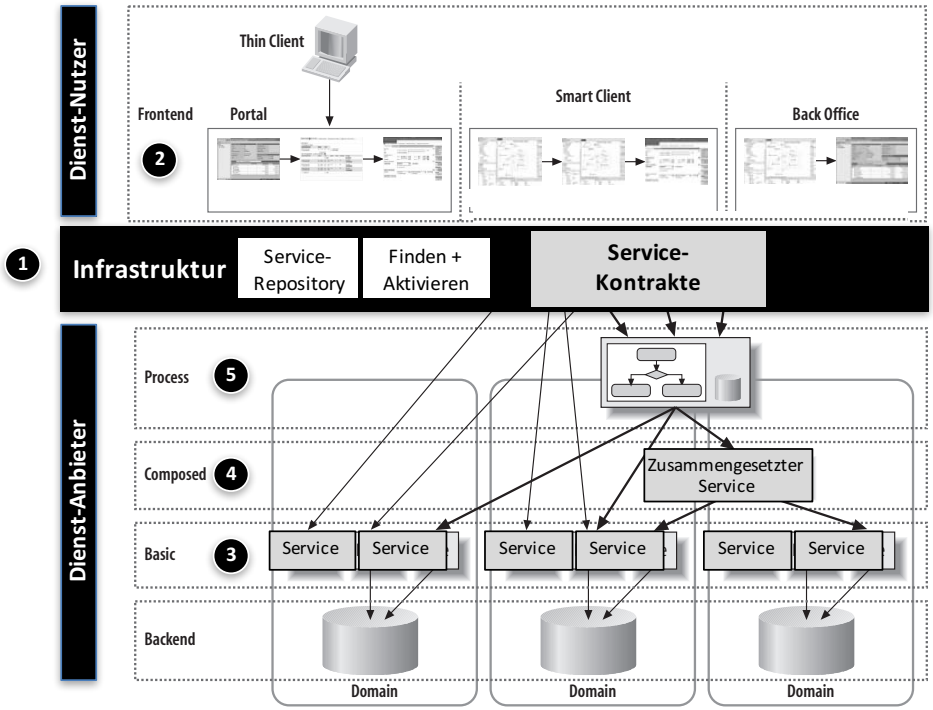


Abbildung 2.1. Konzeptioneller Aufbau einer SOA-Architektur

Die Dienste werden im Rahmen einer **Infrastruktur** bereitgestellt ①, die den Zugriff auf Dienste innerhalb eines Unternehmens ermöglicht. Dienste sind über **Service-Kontrakte** beschrieben, die einerseits deren Funktion und Schnittstelle (Semantik, Ein- bzw. Ausgabeparameter), andererseits deren Nutzungsparameter spezifizieren (z.B. Aufrufadresse eines

[9] Weitere grundlegende Eigenschaften sind in [57, 114, 129] zusammengefasst.

Webservices, Datenformate etc.). Anhand der Kontrakte werden die verfügbaren Dienste in einem Repository verwaltet und Komponenten zum Auffinden und Aktivieren der Dienste bereitgestellt. **Dienst-Nutzer** verwenden die vorhandene Infrastruktur um sich mit Diensten zu verbinden, die eine geforderte Funktion umsetzen. Sie nutzen die im Service-Kontrakt spezifizierte Dienst-Schnittstelle. Die Benutzungsschnittstellen der nutzenden Anwendungen (*frontends*) können dabei mit unterschiedlichen Technologien umgesetzt und für verschiedene Nutzergruppen erstellt sein (z.B. webbasierte oder RichClient-Anwendungen für Endkunden mit Portalzugang oder Backoffice-Mitarbeiter auf ihren Arbeitsplatz-Rechnern ②). Ein **Dienst-Anbieter** stellt Dienste zur Verfügung, welche die Geschäftslogik implementieren. Der Anbieter liefert die Service-Kontrakte, die innerhalb der Infrastruktur und vom Dienst-Nutzer verwendet werden. Die Dienste können elementar sein ③ (*basic*) oder sich aus elementaren Diensten zusammensetzen ④ (*composed*). Darüber hinaus können Teilprozesse als Dienst angeboten werden ⑤ (*process*), die z.B. über Workflow-Systeme orchestriert werden [114].

Die **technische Umsetzung einer SOA** ist grundsätzlich unabhängig von der Verwendung spezifischer Technologien [156]. Dennoch wird SOA häufig mit konkreten Technologien in Verbindung gebracht. Insbesondere sind dies XML-basierte *Web-Service-Technologien* [114, 123], auf deren Basis Anbieter Infrastrukturkomponenten einer SOA aufbauen konnten (z.B. IBM [31], SAP [72], Oracle [46], Microsoft [214]). Sie nutzen eine Sammlung standardisierter Technologien (z.B. SOAP, WSDL, UDDI, HTTP/S), mit denen Dienste beschrieben und die Kommunikation mit Dienst-Nutzern umgesetzt wird [162, 232].

Das SOA-Paradigma löst grundlegende Problemstellungen für die Wiederverwendung von Diensten. Dennoch wuchs in den vergangenen Jahren die Kritik an dessen technischer Umsetzung[10]. So werden in der Praxis vermehrt *RESTful Webservices* [66, 164] und *Microservices* [27] zur Diensterstellung verwendet, die andere Technologie-Stacks verwenden (z.B. JSON statt XML oder REST statt SOAP). Diese Ansätze passen zwar in das grundlegende SOA-Paradigma, können jedoch nicht mit den bestehenden Mitteln integriert werden [29].

2.1.2 Globale Wiederverwendbarkeit in Dienst-Ökosystemen

Auf dem SOA-Paradigma aufbauend entstanden in Wirtschaft und Industrie Initiativen und Konsortien, welche die Normierung von Schnittstellen und Daten für eine Vielzahl an Domänen vorantreiben[11]. Im Versicherungsbereich sind dies beispielsweise für den deutschsprachigen Raum das BiPRO-Konsortium (*bipro.net*) oder für den englischsprachigen ACORD (*acord.org*), im Reisebereich die *Open Travel Alliance* (*opentravel.org*).

Durch die Standardisierung können multiple Dienst-Anbieter Implementierungen für domänenspezifische Dienste anbieten. Das Angebot einer Vielzahl gleichartiger Dienste führt zu sog. **digitalen Dienst-Ökosystemen** [14, 30], die Geschäftsprozesse für spezifische Domänen zusammenfassen. Ein Dienst-Ökosystem wird in Boley et al. [15] folgendermaßen charakterisiert:

[10] http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html

[11] https://www.service-architecture.com/articles/web-services/industry_consortia.html

> ”A service ecosystem is an open, flexible, domain clustered, demand-driven, interactive environment. […] a networked architecture and collaborative environment […]”

Ein Dienst-Ökosystem ist danach eine Menge domänenspezifischer Dienste (*domain clustered*), die von verschiedenen Anbietern (*open*) in einer verteilten Umgebung (*networked architecture*) angeboten und kombiniert werden können (*flexible, collaborative*). Ein Dienst-Ökosystem stellt zudem die technische Umgebung (*environment*) für diese Zusammenarbeit bereit [195]. Ein Dienst-Ökosystem erweitert das SOA-Paradigma, indem es einen Pool kollaborierender domänenspezifischer Dienst-Implementierungen vorsieht [21].

Ein Dienst-Ökosystem stellt erweiterte Anforderungen an die Architektur [65]. Es erfordert eine **ökosystemorientierte Architektur**, in welcher Dienste autonomer Teilnehmer verwaltet werden können [195]. Abbildung 2.2 stellt den konzeptionellen Aufbau eines Dienst-Ökosystems als Erweiterung der in Abbildung 2.1 dargestellten SOA dar. Mehrere Dienst-Anbieter exponieren Implementierungen für Dienste, die von Dienst-Nutzern über standardisierte Dienst-Schnittstellen angesprochen werden können ①. Das Dienst-Ökosystem ist der Vermittler zwischen konkreter Dienstimplementierung und dem Dienst-Nutzer ②.

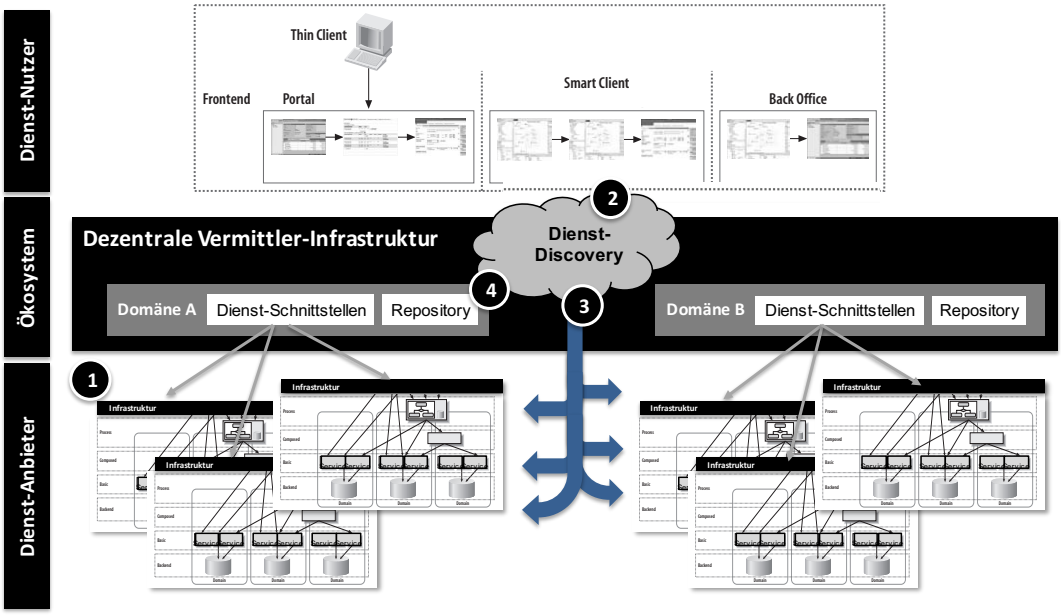


Abbildung 2.2. Konzeptioneller Aufbau eines Dienst-Ökosystems

Der wesentliche Unterschied zu einer SOA besteht darin, dass die lokale Infrastruktur durch eine **dezentrale Vermittler-Infrastruktur** ersetzt wird, welche das Auffinden (*Dienst-Discovery* ③) von Diensten ermöglicht. Sie verwaltet für spezifische Domänen die Dienst-Schnittstellen ④ sowie die Verbindungsinformationen für **multiple Implementierungen**, legt jedoch keine spezifische Implementierung fest.

Um den Einschränkungen einer SOA zu begegnen (vgl. Abschnitt 2.1.1), ist zur **technischen Umsetzung einer ökosystemorientierten Architektur** eine universellere Beschreibung der Dienste und Dienst-Schnittstellen erforderlich. Diese muss eine verteilte Dienst-Discovery erlauben und zum Aufbau einer dezentrale Vermittler-Infrastruktur geeignet sein [30]. Zur Erfüllung dieser Anforderungen werden in den Arbeiten zu Dienst-Ökosystemen **Technologien aus dem *Semantic Web*-Umfeld** verwendet, welche die Beschreibung von Diensten von

einer technischen auf eine semantische Ebene verlagern. Dienste werden über **Ontologien** beschrieben, welche Metadaten zur Klassifikation der Dienste und deren Semantik enthalten sowie deren Dienst-Schnittstellen spezifizieren [15, 50]. Als Grundlage zur Beschreibung der Ontologien dienen insbesondere das *Resource Description Framework* (RDF) [44] bzw. die *Web Ontology Language* (OWL) [94].

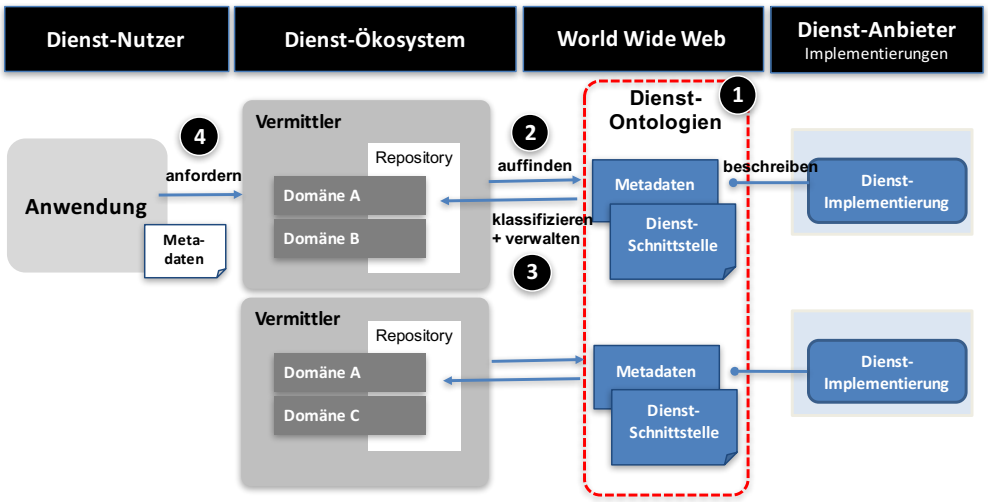


Abbildung 2.3. Nutzung von Ontologien zur Dienst-Beschreibung

Abbildung 2.3 zeigt die Nutzung semantischer Technologien zur Dienstbeschreibung. Anbieter spezifizieren ihre Dienste über **Dienst-Ontologien** ①, welche die Metadaten und Schnittstellen-Daten beschreiben. Diese Informationen werden für Dritte sichtbar publiziert (z.B. auf der Website des Anbieters oder durch explizite Registrierung bei einem Vermittler). **Vermittler** können das Web nach Dienst-Beschreibungen durchsuchen ② (z.B. durch *semantische Web-Crawler* [50]), die gefundenen Dienste aufgrund der Metadaten kategorisieren ③ (z.B. Identifikation der Domäne und Qualitätsstandards) und ihrer Registry hinzufügen. Dienst-Nutzer fragen bei Vermittlern Dienste anhand der Metadaten und ggf. weiterer Merkmale an ④ und erhalten passende Ergebnisse zur Auswahl.

Zur **Beschreibung der Metadaten von Diensten** sowie deren Auswahl existieren bereits Standards, z.B. *OWL-S* (Web Ontology Language for Web Services) [136], *WSMO* (Web Service Modeling Ontology) [54] und *SAWSDL* (semantic annotations for WSDL) [62], die sich weitgehend an den Inhalten der *Web Service Description Language* orientieren. Ergänzend existieren Ansätze zur optimierten Auswahl von Diensten. So fokussieren z.B. *DIANE* [125] und *Easy-L* [145] und die Arbeiten von Barkat et al. [12] und Hang et al. [86] auf eine effiziente Klassifizierung und Auswahl von Diensten.

Potential von Dienst-Ökosystemen für die Zukunft

Hinsichtlich der **Standardisierung von Dienst-Schnittstellen** sind Bestrebungen der Industrie vorhanden, bestehende Standards zur Beschreibung von Geschäftsprozessen auf Semantic Web Technologien zu adaptieren. So kündigte z.B. das BiPRO-Konsortium für den Versicherungsbereich die Verwendung von Semantic Web-Standards als Ergänzung zu bestehenden Normen an[12]. Im Automotive-Bereich arbeitet Volkswagen an Ontologien[13], die Fahr-

[12] https://www.bipro.net/bipro_tag_2016/der_bipro_tag_2016

[13] http://www.dataversity.net/volkswagen-das-auto-company-is-das-semantic-web-company/

zeuge und Komponenten beschreiben (*Volkswagen Vehicles Ontology*[14], *Car Options Ontology*[15]). Im akademischen Bereich existieren beispielhafte Vokabularien und Ontologien zur Beschreibung thematisch abgegrenzter Bereiche[16], z.B. für Personendaten, Orte, Veranstaltungen, Aktivitäten oder zur Beschreibung von e-Commerce Gütern und Dienstleistungen über die *GoodRelations*-Ontologie[17].

Die Vorteile von auf semantischen Technologien basierenden Dienst-Ökosystemen liegen jedoch nicht nur in der technischen Anbindung von Diensten an die IT-Infrastruktur eines Unternehmens. Sie bieten auch das Potential zur Umsetzung innovativer Geschäftsmodelle, die durchaus *disruptiven* Charakter [20, 37] aufweisen können. Im Sinne des *Shared Economy*-Konzepts [17, 32, 127, 200] können über Dienst-Ökosysteme digitale Marktplätze (*Digital Marketplaces* [131]) unabhängig von etablierten Anbietern aufgebaut werden und neuartige Markträume entstehen (*Distributed Marketspaces* [183, 185]).

2.2 Dialogbasierte Anwendungen

Geschäftsprozesse benötigen zur Durchführung Eingabedaten, die vom Nutzer bereitgestellt werden müssen. Insbesondere die Integration menschlicher Nutzer in den Prozess erfordert Benutzungsschnittstellen zu deren Erfassung. Ein häufig auftretendes Anwendungsmuster sind **dialogbasierte Anwendungen** (*form filling* oder *directed dialog* [34]). Sie sind eine Form des angeleiteten Dialogs zur strukturierten Erfassung von Informationen, die ein Geschäftsprozess benötigt. Die Durchführung eines Geschäftsprozesses besteht ggf. aus mehreren aneinandergereihten Schritten (vgl. [34]). Beispiele sind eine Überweisung, die Buchung eines Fluges, einer Übernachtung oder der Antrag für ein Versicherungsprodukt. In vertriebsnahen Systemen wird dieses Anwendungsmuster häufig verwendet und besitzt dort eine hohe Relevanz. So folgen z.B. in Finanz- und Versicherungsportalen ca. 65%-95% der Anwendungen diesem Muster (s. Anhang A.1).

Abbildung 2.4 zeigt das Prinzip dialogbasierter Anwendungen. Dem Nutzer wird eine Benutzungsschnittstelle angeboten, die in einer Folge von Schritten die Eingabedaten für einen Prozess erfasst (z.B. *Kundendaten* und *Produktkonfiguration* mit Detailfragen). Diese werden anschließend einem Dienst zur Verarbeitung übergeben. Ergebnisse der Verarbeitung werden dem Nutzer angezeigt (z.B. als *Angebot*) und ggf. folgen weitere Schritte, die Daten erfassen und an Dienste übergeben.

Die Benutzungsschnittstellen dialogbasierter Anwendungen sind eingabeorientiert und weisen einen formularartigen Charakter auf. Sie sind in hohem Maße **datengetrieben**, ermöglichen eine strukturierte Eingabe und weisen ein dynamisches Verhalten auf, welches die Erfassung unterstützt. So variiert z.B. abhängig von bereits erfassten Informationen der Fragefluss und passt sich dynamisch der Datenlage an. Sie grenzen sich dadurch von **ausgabeorientierten Anwendungen** ab, deren Ziel die Aggregation und Darstellung von Informationen ist (z.B. Vertrags- oder Kontoübersichten, Portal-Einstiegsseiten), die einen vergleichsweise statischen Charakter aufweisen.

[14] http://purl.org/vvo/ns

[15] http://purl.org/coo/ns

[16] http://schema.org/docs/schemas.html

[17] http://purl.org/goodrelations/v1

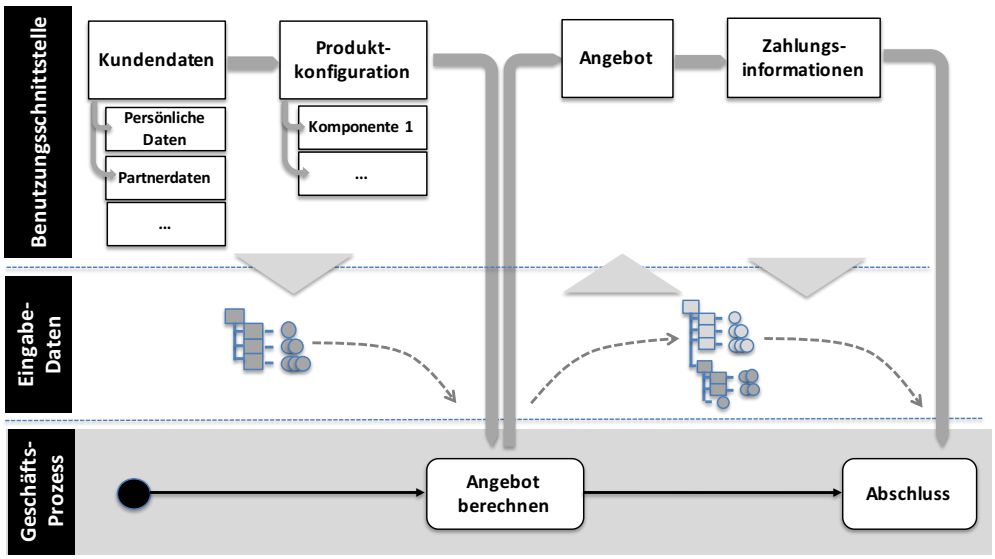


Abbildung 2.4. Prinzip dialogbasierter Anwendungen

2.2.1 Beispiel: Antragstrecke Haftpflichtversicherung

Das folgende Beispiel illustriert den datengetriebenen Charakter dialogbasierter Anwendungen am Beispiel einer grafischen Benutzungsschnittstelle für den Antrag zu einer privaten Haftpflichtversicherung.

Abbildung 2.5 zeigt exemplarisch zwei Seiten der Antragsstrecke, die zur Nutzung durch Vermittler konzipiert wurden. Die Antragstrecke erfasst sukzessive alle für den Antragsprozess relevanten Informationen im Dialog mit dem Benutzer. Hierzu werden die Daten in Fragebereichen erfasst, die sukzessive beantwortet werden und anschließend zur Berechnung des Angebots führen.

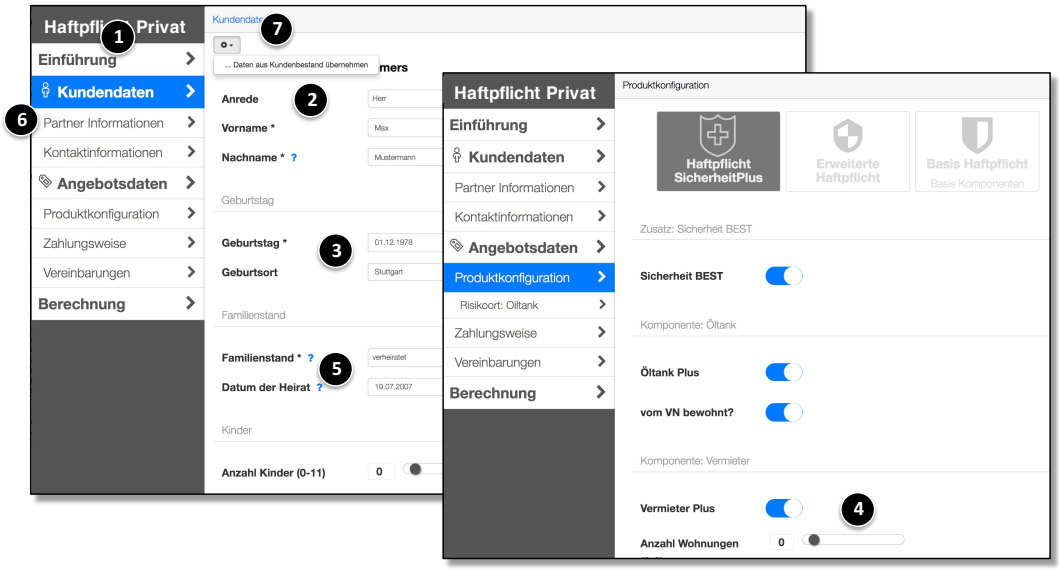


Abbildung 2.5. Antragstrecke Haftpflichtversicherung aus Vermittlersicht

Die Präsentation der Daten erfolgt strukturiert in **Seiten und Unterseiten**, die in einer hierarchischen Beziehung zueinander stehen und über eine Navigation wählbar sind ① (z.B.

Kundendaten, *Partnerinformationen*, *Kontaktinformationen* und *Angebotsdaten*). Die Daten einer Seite werden wiederum in **Gruppen und Untergruppen** erfasst (z.B. *Namensinformationen*, *Geburtstag*, *Familienstand* und *Kinder* ②). Die Eingaben werden in einer für den Nutzer nachvollziehbaren **Reihenfolge** präsentiert (z.B. *Vorname* vor *Nachname*). Die Erfassung erfolgt über **Interaktionselemente**, die abhängig von der Art des zu erfassenden Datums variieren (z.B. Textfelder, Datumseingabefelder ③, Schieberegler oder Auswahlfelder ④).

Die Anwendung besitzt darüber hinaus ein **dynamisches Verhalten**, das von bereits erfassten Daten abhängt. Dies reicht von der Vorbelegung voneinander abhängiger Inhalte (z.B. Ort abhängig von der Postleitzahl) bis hin zu Existenzbedingungen, die den Fragefluss beeinflussen (z.B. Anzeige des *Datums der Heirat* ⑤ und der *Partnerinformationen* ⑥, falls der Nutzer den *Familienstand verheiratet* auswählt). Weitere Funktionalitäten sind die Validierung von Eingaben und ein entsprechendes Fehlerverhalten, sowie Aktionen, die der Nutzer explizit über Schaltflächen oder Menüs initiiert (z.B. Öffnen einer Kundendatenbank zur Vorbelegung von Kundendaten ⑦).

2.2.2 Eigenschaften dialogbasierter Benutzungsschnittstellen

Qualitativ hochwertige Benutzungsschnittstellen zeichnen sich dadurch aus, dass sie den Anwender bedarfsgerecht während der Bedienung leiten und unterstützen [107, 151, 172]. Bei der Erstellung werden Gestaltungs- und Interaktionsmuster angewendet [89], die zu einer nutzergerechten Schnittstelle führen. Das Aussehen und das Verhalten variieren dabei anhängig von der Nutzungsumgebung [25, 110]. Die Umsetzung erfolgt mittels spezifischer Technologien, die auf diese Umgebung zugeschnitten sind und sich in den Interaktionsformen unterscheiden (z.B. grafische Eingabe über Maus, Tastatur oder Stift, Spracherkennung, taktile Eingaben oder manuelle Gesten) [160]. Dies wird als **Modalität einer Anwendung** bezeichnet [25, 124]. Die Vielzahl möglicher Interaktionsmuster erschwert oder verhindert sogar die generelle Generierbarkeit beliebiger Benutzungsschnittstellen [58, 170, 180, 218].

Dialogbasierte Benutzungsschnittstellen weisen jedoch durch ihren datengetriebenen, formularartigen Charakter **spezifische Eigenschaften** auf, die auf alle Modalitäten anwendbar sind. Sie präsentieren eine Sequenz gruppierter Fragen, verwenden für die Beantwortung typabhängige Interaktionselemente und besitzen ein von bereits erfassten Daten abhängiges Verhalten. Chlebek [34] charakterisiert dialogbasierte Benutzungsschnittstellen als *"[...] die Summe der Bedienelemente, deren Anordnung und Wechselwirkung"*. Danach können dialogbasierte Benutzungsschnittstellen über folgende Eigenschaften beschrieben werden:

- Struktur und Aufbau
- Typisierte Interaktionselemente zur Ein-/Ausgabe
- Verhalten der Benutzungsschnittstelle

Dies sind elementare Eigenschaften, für deren praktische Umsetzung alle Modalitäten Lösungen vorsehen. In allen gängigen Technologien existieren Möglichkeiten zur Strukturierung und typisierten Erfassung von Daten sowie Konzepte zur Realisierung dynamischen Verhaltens bei der Eingabe.

Eine weitere Eigenschaft ist der **hohe Standardisierungsgrad** dialogbasierter Benutzungsschnittstellen. Zu deren Gestaltung wurden in den vergangenen Jahren grundlegende Re-

geln geschaffen (z.B. [48, 89, 107, 177]) und allgemeine Standards erarbeitet (z.B. DIN EN ISO 9241-110: Dialog-Prinzipien [107], DIN EN ISO 9241-143: Formularbasierte Dialoge [108]). Diese werden um Gestaltungsrichtlinien für spezifische Technologien ergänzt, die eine einheitliche Nutzererfahrung (*User Experience*) für eine Vielzahl von Modalitäten gewährleisten (z.B. [222, 224]). In Unternehmen kommen *Styleguides* [217] hinzu, welche die Freiheitsgrade bei der Erstellung der Benutzungsschnittstellen weiter eingrenzen.

Aufgrund ihrer spezifischen Eigenschaften und des hohen Standardisierungsgrades eignen sich dialogbasierte Benutzungsschnittstellen für eine automatisierte Erstellung.

2.3 Varianten dialogbasierter Anwendungen

In Dienst-orientierten Umgebungen besteht der Bedarf zur Anbindung einer potentiell großen Zahl unterschiedlicher Nutzergruppen an einen Dienst, die über verschiedene Zugangswege mit dem System interagieren. Dies erfordert Varianten von Benutzungsschnittstellen, welche die Funktionalität der Anwendung auf eine spezifische Nutzungsumgebung angepasst präsentieren [160]. Anwendungen, die multiple Nutzungsumgebungen unterstützen, werden von Calvary et al. [25] als **multimodale Anwendungen** (*multi-context, multi-target user interfaces*) bezeichnet.

Die Varianten unterscheiden sich abhängig von der Nutzungsumgebung. Sie hängen von den Anforderungen und Fähigkeiten der Nutzer ab [107] und werden unter dem Gesichtspunkt einer optimierten Nutzbarkeit erstellt. Die Norm DIN EN IS0 9241-11 [110] definiert Nutzbarkeit (*Usability*) als

> *" [...] extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*

Die Nutzbarkeit wird danach von zwei Aspekten bestimmt, die die Erstellung von Varianten beeinflussen:

- Zuschnitt auf eine Nutzergruppe und deren Anforderungen
- Technische Umsetzung, die den Nutzer effizient und effektiv unterstützt

Varianten einer Anwendung sind damit funktional gleichartige dialogbasierte Anwendungen, die inhaltlich auf eine spezifische Nutzergruppe zugeschnitten und für eine spezifische Technologie optimiert sind.

2.3.1 Beispiel: Varianten der Antragstrecke Haftpflichtversicherung

Die folgenden Beispiele illustrieren die nutzerspezifischen und technischen Unterschiede an Varianten der Antragstrecke Haftpflichtversicherung aus Abschnitt 2.2.1. Abbildung 2.6 zeigt eine webbasierte Variante, die für **Endkunden im Unternehmensportal** bestimmt ist. Der Umfang der Fragen ist im Vergleich zur Vermittler-Variante deutlich reduziert. Es werden lediglich grundlegende Informationen zum Kunden abgefragt (z.B. *Geburtsdatum*, *Familienstand*, *Kinder im Haushalt* ①). Eine weitere Detaillierung erfolgt hier nicht (z.B. keine Erfassung der Partnerdaten). Auch die Auswahl der Produktkomponenten ist vereinfacht.

Hier werden keine weiteren Nachfragen z.B. zur Adresse der Wohnungen bei Auswahl der *VermieterPlus*-Komponente gestellt ②.

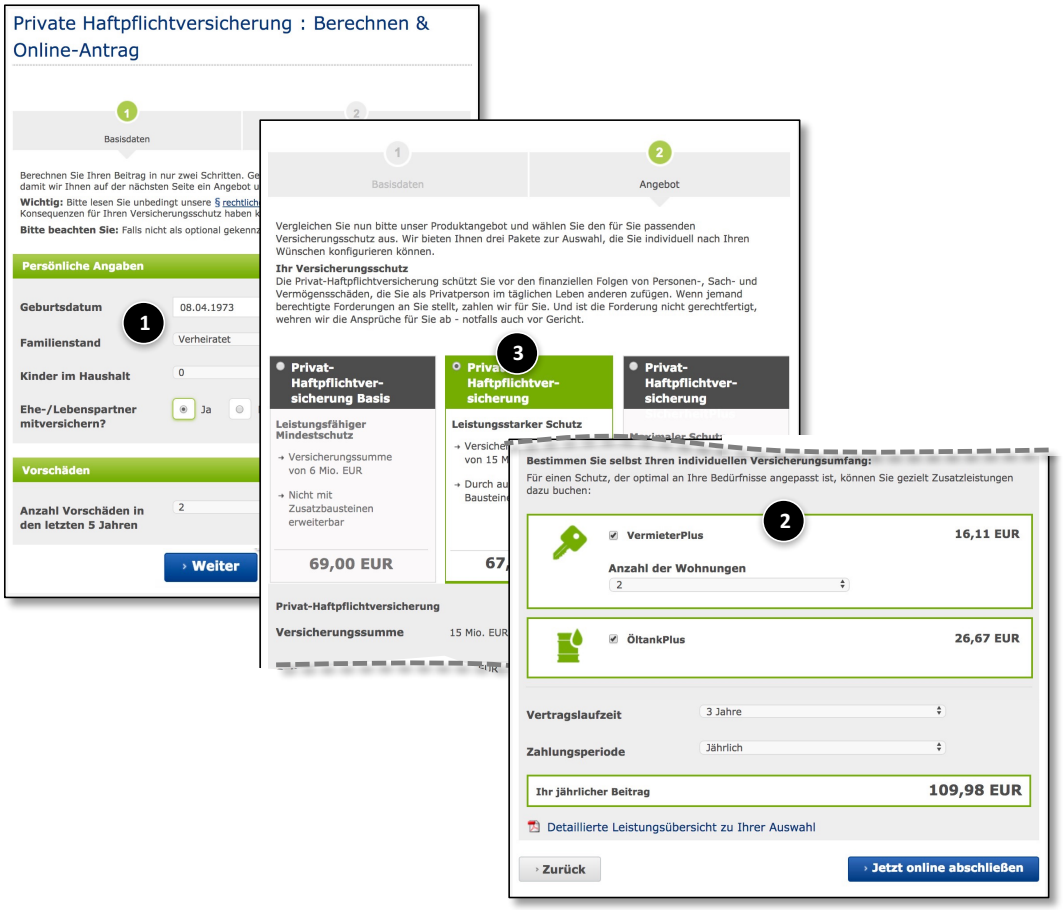


Abbildung 2.6. Antragstrecke Haftpflichtversicherung (Kundensicht / web-Variante)

Die Anwendung ist auf zwei Seiten verteilt. Die Navigation erfolgt sequentiell über eine *Weiter* und *Zurück*-Schaltfläche. Aufgrund des verminderten Umfangs der erfassten Daten werden Gruppen in der Darstellung vereinfacht. So entfallen z.B. die Überschriften der Gruppen Geburtstag und Familienstand aus der Vermittler-Variante, da hier lediglich ein Feld der Gruppe dargestellt wird ①.

Aufgrund der Plattform und dort verwendeten Technologie, unterscheidet sich das Erscheinungsbild erheblich von der Vermittler-Variante. Es werden auf der Plattform verfügbare Interaktionselemente verwendet, die z.T. auf die Nutzergruppe zugeschnitten sind. So erfolgt die Auswahl des Produkts ③ über ein spezielles Interaktionselement, das mehr Informationen zum Produkt anzeigt als in der Vermittler-Variante. Textuelle Informationen (Labels, Hilfetexte, Erläuterungen) sind in dieser Variante umfangreicher als in der Vermittler-Variante und auf Endkunden zugeschnitten.

Abbildung 2.7 zeigt eine Variante, die für **Endkunden zur Nutzung auf einem mobilen Gerät** bestimmt ist. Das Ziel dieser Variante ist es, eine möglichst niedrige Hürde für den Abschluss des Produkts zu legen. Hierzu wird das Produkt nur in einer reduzierten Version angeboten (z.B. Verzicht des Angebots der Zusatzkomponenten Vermieter- und Öltank-Zusatzversicherung) und damit inhaltlich noch weiter eingeschränkt. Bei den *Kundendaten*

wurde z.B. zusätzlich auf die Angabe der Geburts- und Zusatzinformationen zu Heirat und Kindern verzichtet. Die *Kontaktdaten* wurden auf den Online-Kontakt reduziert und im Fragefluss der Gruppe *Kundendaten* hinzugefügt.

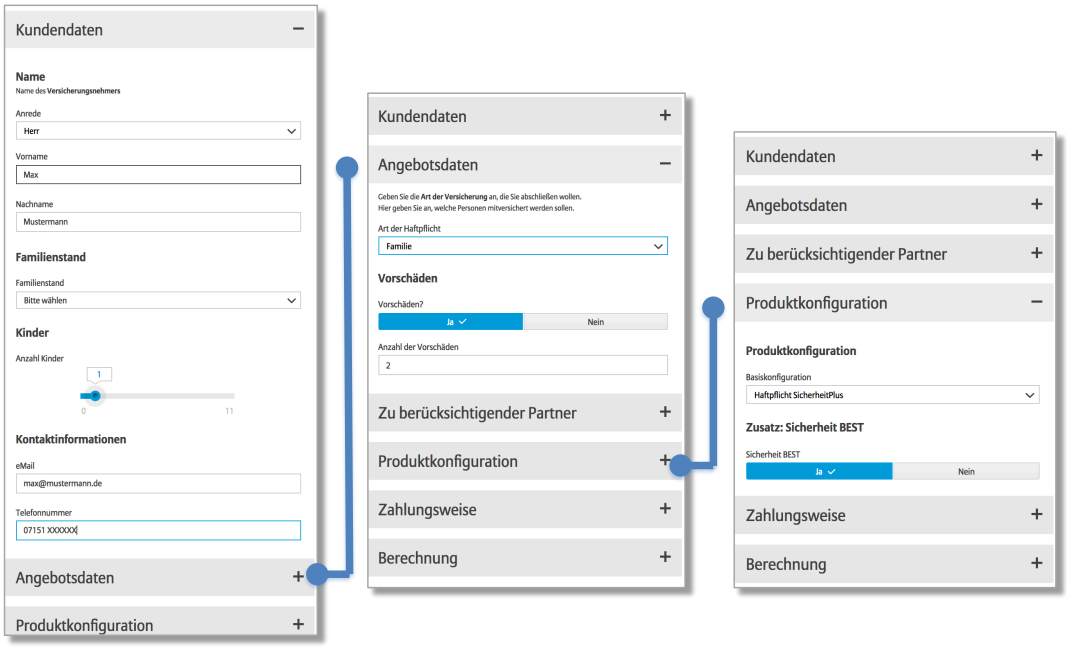


Abbildung 2.7. Antragstrecke Haftpflichtversicherung (Kundensicht / mobile-Variante)

Durch die Reduktion wurde den räumlichen Restriktionen mobiler Endgeräte Rechnung getragen. Die Navigation erfolgt sequentiell, gestattet jedoch den freien Zugriff auf Fragegruppen, die als aufklappbare Bereiche angeboten werden. Die baumartige Navigationsstruktur der Vermittler-Variante wird aufgelöst und Fragegruppen in den Fragefluss der übergeordneten Gruppe integriert dargestellt. Das Erscheinungsbild ist auf mobile Endgeräte zugeschnitten. Die Interaktionselemente sind stark vereinfacht. Die textuellen Informationen sind auf Endkunden zugeschnitten, jedoch im Umfang reduziert.

2.3.2 Einflussfaktoren für Varianten

Die o.g. nutzerspezifischen und technischen Unterschiede können als voneinander unabhängige Aspekte betrachtet werden. Im Folgenden wird zwischen **inhaltlichen** und **technischen** Varianten unterscheiden.

Inhaltliche Varianten ergeben sich aus den Anforderungen der Nutzergruppe, welche die Variante nutzt (z.B. fachliche Ausrichtung, vorhandenes Hintergrundwissen, Nutzungsumgebung, physische Einschränkungen der Nutzer). Bei dialogbasierten Anwendungen äußert sich dies in einem angepassten Umfang der zu erfassenden Daten (z.B. reduzierte Zahl von Fragen für Endkunden- im Vergleich zu Vermittler-Varianten) und in einem an die Nutzergruppe angepassten Verhalten.

Technische Varianten ergeben sich aus den technischen Rahmenbedingungen, in denen die Anwendung eingesetzt wird. Sie sind bestimmt durch das Interaktionsmedium (z.B. Ein-/Ausgabegerät), Plattformspezifika und verfügbare Technologien, die zur Umsetzung vorgegeben sind (z.B. webbasierte, mobile, desktopbasierte oder sprachgesteuerte Anwendungen).

Diese bestimmen die Präsentation und das Verhalten der Anwendung (z.B. verwendbare Interaktionselemente, deren Verteilung auf Seiten und die Navigation).

Die Erstellung von Varianten hängt von der Nutzung einer Anwendung ab. Dies wird in [25] unter dem Begriff *Context-Of-Use* zusammengefasst und folgende Einflussfaktoren angeführt, die die Erstellung von Varianten bestimmen:

- **Nutzer**, die das System effektiv verwenden
- **Physische Umgebung**, in der die Interaktion stattfindet
- **Hardware- und Software-Plattformen**, die zur Interaktion verwendet werden

Zur Unterscheidung der Einflussfaktoren kann der *Context-of-Use* in die Bereiche **Nutzungskontext**, **Interaktionskontext** und **Technologiekontext** untergliedert werden.

Der **Nutzungskontext** beschreibt Aspekte, die sich aus der Nutzung der Anwendung ergeben. Insbesondere beinhaltet er Informationen zu folgenden Bereichen, aus denen Eigenschaften einer Benutzungsschnittstelle abgeleitet werden:

- **Nutzergruppe**: Einflüsse, die nutzergruppenspezifische und fachlich bedingte Eigenschaften betreffen. Diese sind fachlich motiviert (z.B. Detaillierungsgrad der zu erfassenden Daten, fachliche Kenntnisse, Fähigkeiten und Berechtigungen des Nutzers). Hinzu kommen ggf. vorhandene Einschränkungen der Nutzer, z.B. physische Herausforderungen, die alternative Eingaben erfordern (z.B. angepasste Inhalte zur Umsetzung eines barrierefreien Zugangs[18])

- **Umfeld**: Einflüsse der Arbeitsumgebung, in welcher die Anwendung eingesetzt wird, z.B. in einer Büroumgebung, im Außenbereich oder mobilen Umfeld.

- **Lokation**: Einflüsse des geografischen Ortes, in welchem die Anwendung eingesetzt wird, z.B. Sprache und länderspezifische Eingaben (z.B. Postleitzahlen) oder kulturelle Unterschiede in Sprache, Optik und Symbolik.

Der Nutzungskontext beeinflusst den Detaillierungsgrad der Benutzungsschnittstelle und deren nutzungsabhängige Darstellung (z.B. angebotene Labels, unterstützende Hilfetexte und grafische Darstellungen).

Der **Interaktionskontext** liefert Informationen zur physischen Umgebung. Insbesondere sind dies Eigenschaften des technischen Interaktionsmediums [25, 28, 36]. Sie bestimmen die Darstellungs- und Interaktionsrestriktionen, die für ein Medium gelten (z.B. für Bildschirm-, Sprach-, Tastatur-, Maus-, Touch-Screen-, Gesten-Interaktion).

Der **Technologiekontext** bestimmt die Umsetzung der Benutzungsschnittstelle mit konkreten, auf der Zielplattform verfügbaren Technologien. Er legt fest, mit welchen Interaktionselementen Daten präsentiert bzw. erfasst werden und wie das Verhalten der Anwendung für einzelne Gerätekategorien umgesetzt wird (z.B. mittels HTML5/Javascript-Frameworks für webbasierte, Java/JavaFX für Desktop- oder iOS und Android für native Smartphone-Anwendungen).

[18] https://www.w3.org/TR/WCAG20/

Der Nutzungskontext bestimmt maßgeblich die Erstellung inhaltlicher Varianten. Hieraus werden die darzustellenden Informationen und das Verhalten abgeleitet, das für eine spezifische Nutzergruppe in einem bestimmten fachlichen Umfeld benötigt wird. Der Interaktionskontext trägt zur Erstellung sowohl inhaltlicher als auch technischer Varianten bei. Er bestimmt deren mögliche Komplexität, den strukturellen Aufbau und verwendbare Navigationsformen. Der Technologiekontext bestimmt die Umsetzung der technischen Varianten für eine konkrete Plattform.

2.4 Komposition von Anwendungen

In kundennahen Anwendungen treten Aufgabenstellungen auf, die wiederkehrend in Geschäftsprozessen benötigt werden. Insbesondere für Aufgabenstellungen, die in sich abgeschlossene Teilprozesse durchführen, existieren häufig bereits Benutzungsschnittstellen, die als eigenständige Anwendungen angeboten werden und sich wiederverwenden lassen (z.B. zur Änderung von Adressdaten, des Namens bei einer Heirat oder der Bankverbindung). Zur Vermeidung von Mehrfachentwicklungen ist daher eine häufig gestellte Forderung aus der Praxis, dass bestehende Anwendungen als Komponenten **in neuen Anwendungen wiederverwendet** und ggf. der **neuen Nutzung angepasst** werden können.

Ein Anwendungsfeld ist das **kombinierte Angebot digitaler Dienstleitungen eines Unternehmens**, das sich an der konkreten Situation eines Kunden orientiert. Hierzu werden Dienste in sog. *User Journeys*[19] (auch *Customer Journeys* [19, 191]) zusammengestellt und der Kunde bedarfsgerecht durch Anwendungsfälle geführt. Die Kombination ergibt dabei aus Kundensicht **funktional höherwertige Anwendungen**. Ein Beispiel aus dem Versicherungsbereich ist die Änderung des Familienstandes eines Kunden, die eine Änderung der persönlichen sowie von Vertragsdaten (z.B. Bezugsrechte, versicherte Personen) nach sich zieht. Statt den Kunden diese Änderungen einzeln durchführen zu lassen, werden in einer *User Journey* die relevanten Dienste in einer kombinierten Anwendung zusammengefasst. Redundante Eingaben werden dabei vermieden (z.B. Änderung der Anschrift im Kundenkonto sowie in allen Verträgen), was zu einem verbesserten Nutzererlebnis führt.

Ein weiteres Anwendungsfeld ist die **Integration von Dienstleistungen externer Anbieter** aus unterschiedlichen fachlichen Domänen. So entstanden in den vergangenen Jahren vermehrt Anbieter, die kombinierte Dienstleistungen anbieten (z.B. Reiseanbieter, die Flug-, Hotel- und Mietwagen-Buchung vereinen) oder Dienste zur Integration anbieten (z.B. *paypal.com*, *sofortüberweisung.de*, *paydirekt.de*, *trustedshops.de*, *billpay.de*).

2.4.1 Beispiel: Reisebuchung mit Reiseversicherung

Im Folgenden wird die Kombination von Anwendungen unterschiedlicher Domänen am Beispiel einer Reisebuchung mit optionaler Reiseversicherung illustriert. Abbildung 2.8 zeigt exemplarisch die Abschlussseite der Buchung (exemplarisch an Reisebuchung auf *tui.com* angelehnt). Der Kunde hat bereits eine Hotelunterbringung und einen Flug ausgewählt. Zudem wurden bereits die *Persönlichen Reisedaten* sowie die *Rechnungsdaten* erfasst. In der Rubrik *Reiseschutz* wird die Reiseversicherung eines Fremdanbieters angeboten, die als Komponente integriert ist ①.

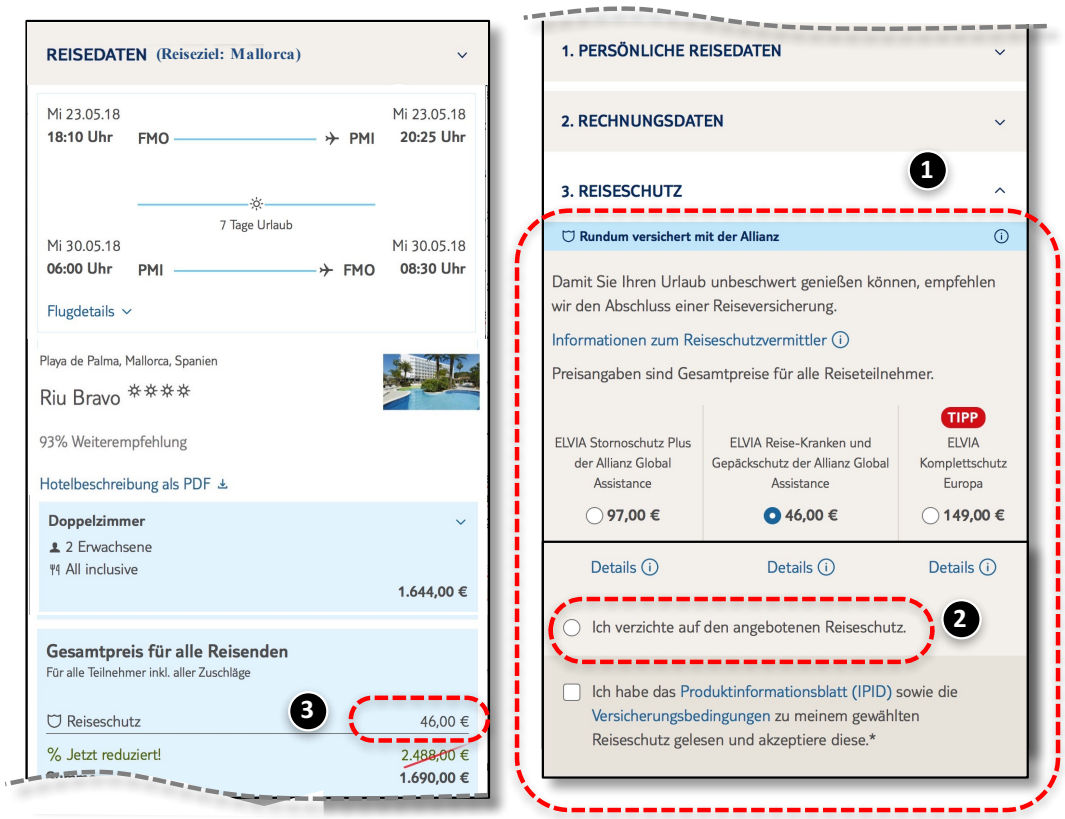


Abbildung 2.8. Kombinierte Reisebuchung mit Reiseversicherung

Die Versicherungs-Komponente ist eine selbständig lauffähige Anwendung, die in der ursprünglichen Benutzungsschnittstelle alle Informationen zum Abschluss der Versicherung erfasst (z.B. Kunden-, Rechnungsdaten und Anzahl der Reisenden). Da diese bereits in der Rahmenanwendung bekannt sind, wurden diese aus der kombinierten Benutzungsschnittstelle entfernt (inhaltliche Modifikation). In die Komponente wurde eine Checkox eingefügt, mit welcher die Versicherung abgewählt werden kann ②. Die Komponente interagiert mit der umgebenden Anwendung und beeinflusst deren Verhalten (Verhaltensmodifikation). Sobald der Kunde eine Produktvariante auswählt, wird der Preis den Reisekosten aufgeschlagen und die Summe in der Rahmenanwendung angezeigt ③.

Die dargestellte Komposition führt zu einer **modifizierten Nutzung der Komponente**. Diese erfolgt unabhängig vom Ersteller der ursprünglichen Komponente und konnte zum Erstellungszeitpunkt nicht vorhergesehen werden.

2.4.2 Eigenschaften von Komponenten in Kompositionen

Die Wiederverwendung existierender Benutzungsschnittstellen besteht aus technischer Sicht in der Integration von Komponenten in eine Rahmenanwendung. Unter einer Komponente wird eine bestehende Anwendung verstanden, die eine in sich abgeschlossen Funktionalität umsetzt und die bereits über eine Benutzungsschnittstelle verfügt. Bei deren Integration

[19] http://www.netzstrategen.com/sagen/webwissen-user-journeys/

können zwei grundlegende Vorgehensweisen unterschieden werden: die **Aggregation** und die **Komposition**[20].

Bei der **Aggregation** wird eine Komponente eingebunden, ohne diese zu modifizieren. Die Komponente ist Teil der Rahmenanwendung, arbeitet jedoch als selbständige Einheit unabhängig vom umgebenden Kontext. Bei der **Komposition** werden die Komponenten integraler Bestandteil der Rahmenanwendung und werden dabei an den spezifischen Kontext der Wiederverwendung angepasst. Die Komponenten müssen hierzu folgende Eigenschaften aufweisen:

- Sie stellen selbständig funktionsfähige Komponenten dar
- Nutzbarkeit an beliebigen Stellen im Fragefluss der Rahmenanwendung
- Modifizierbarkeit des Inhalts und Verhaltens abhängig von der Nutzung
- Nutzbarkeit in verschiedenen Interaktions- und Technologiekontexten

Kompositionen ermöglichen eine **nahtlose Integration bestehender Anwendungen**. Die Komponenten werden an beliebigen Stellen eingefügt und interagieren mit der Rahmenanwendung (vgl. Abschnitt 2.4.1). Hierzu werden sie ggf. für die Zusammenarbeit mit der Rahmenanwendung modifiziert. Die Darstellung der Komponenten muss sich dazu dem Interaktions- und Technologiekontext der Rahmenanwendung anpassen.

2.5 *Shared UIs*: Wiederverwendung in Dienst-Ökosystemen

Dienste eines Dienst-Ökosystems setzen in sich abgeschlossene Teilprozesse um und werden von verschiedenen Anbietern unabhängig von einer spezifischen Nutzung bereitgestellt (vgl. Abschnitt 2.1.2). Dienste werden dabei unabhängig von Benutzungsschnittstellen erstellt. Gängige Praxis zur Integration der Dienste in eigene Anwendungen ist daher die manuelle Erstellung oder Nutzung von einem Dienst-Anbieter bereitgestellter Benutzungsschnittstellen, die auf den konkreten Nutzungs-, Interaktions-, Technologiekontext zugeschnitten sind. Die Anbieter können jedoch nicht vorhersehen, in welchem Umfeld der Dienst verwendet wird. Die Benutzungsschnittstellen sind damit nicht beliebig wiederverwendbar.

Die Wiederverwendung von Benutzungsschnittstellen als Kompositionen (vgl. Abschnitt 2.4) kann insbesondere in Dienst-Ökosystemen einen Mehrwert bei der Entwicklung neuer Anwendungen schaffen. Die hier gegebene Standardisierung der Dienst-Schnittstellen (vgl. Abschnitt 2.1.2) ermöglicht die vom Dienst-Anbieter entkoppelte Erstellung von Benutzungsschnittstellen. Dienst-Schnittstellenbeschreibungen enthalten eine umfassende Spezifikation der zu erfassenden Daten. Darauf basierend können von beliebigen Anbietern Benutzungsschnittstellen erstellt werden, die zur Spezifikation passende Daten erfassen und an beliebige Dienst-Implementierungen angebunden werden können. Erfüllen diese Benutzungsschnittstellen die in Abschnitt 2.5.2 aufgeführten Komponenteneigenschaften, sind sie von Nutzern i.S. einer Komposition integrierbar und können an ein neues Umfeld angepasst werden.

[20] Die Begriffe Aggregation und Komposition wurden in Anlehnung an entsprechende Konzepte der Objektmodellierung gewählt. Vgl. z.B. [196], S. 152ff.

Durch die Standardisierung der Daten entsteht so eine **gemeinschaftliche Nutzbarkeit** einmal erstellter Benutzungsschnittstellen. Diese werden im Folgenden als **Shared UIs**[21] bezeichnet. Hierunter werden frei verfügbare Benutzungsschnittstellen verstanden, die als eigenständig funktionierende Komponenten in Anwendungen integriert, angepasst, sowie an verschiedene Dienst-Implementierungen angebunden werden können. Für ein Ökosystem-Szenario ergeben sich daraus neue Möglichkeiten der Wiederverwendung.

2.5.1 Beispiel: Komposition der Reisebuchung aus *Shared UIs*

Abbildung 2.9 zeigt die Umsetzung des in Abschnitt 2.4.1 dargestellten Szenarios einer Reisebuchung mit *Shared UIs*. Die Reisebuchung ist eine Komposition von *Shared UIs* für die Flugbuchung ①, Hotelbuchung ② und Reiseversicherung ③ in einer Rahmenanwendung. Diese wird um die Erfassung übergreifender Informationen (*Persönliche Reisedaten*, *Rechnungsdaten* ④) sowie die Darstellung abgeleiteter Informationen (z.B. Berechnung der Gesamtkosten ⑤) erweitert. In der Abbildung ist die Angabe der Flug- und Hoteldaten bereits erfolgt und die Komponenten zeigen das Ergebnis der Auswahl. Die Darstellung der Anwendung unterscheidet sich nur geringfügig vom Beispiel aus Abschnitt 2.4.1. Lediglich die Reiseversicherungs-Komponente ③ ist anbieterunabhängig gestaltet.

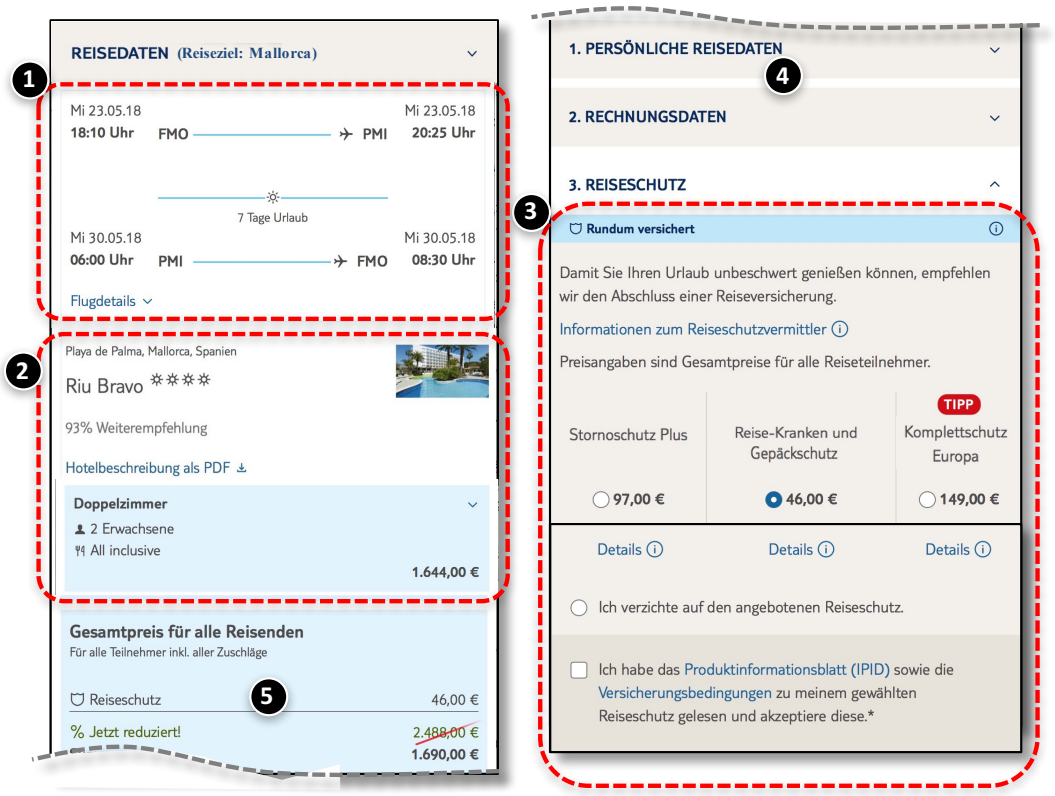


Abbildung 2.9. Umsetzung der Reisebuchung mit *Shared UIs*

[21] In der Arbeit wird der englische Begriff *sharing* verwendet, da er viele Aspekte der gemeinsamen Nutzung abdeckt, die in der deutschen Sprache nicht in einem Wort zu erfassen sind. *Sharing* bedeutet u.a. *teilen, teilhaben, gemeinsam haben, teilnehmen, gemeinsam tragen, sich etw. teilen, sich an etw. beteiligen, an etw. beteiligt werden*. Er beinhaltet sowohl passive als auch aktive Aspekte der Zusammenarbeit (*Nutzung von und Beitrag zu Gemeinschaftsleistungen*).

Der grundsätzliche Unterschied zum Beispiel in Abschnitt 2.4 ist technischer Natur. Er besteht darin, dass *Shared UIs* als eigenständige Komponenten eingebunden werden, die mit standardisierten Diensten eines Ökosystems interagieren. Es stellt eine Erweiterung des Kompositionsansatzes dar, da hier von konkreten Dienst-Anbietern unabhängig erstellte Komponenten genutzt werden. Sie basieren auf standardisierten Dienst-Schnittstellen des Ökosystems und können an beliebige Dienstimplementierungen angebunden werden.

2.5.2 Prinzip und Eigenschaften von *Shared UIs*

Zur Verdeutlichung der Unterschiede zeigt Abbildung 2.10 das Prinzip von *Shared UIs* aus technischer Sicht. Sie stellt die Zusammenhänge zwischen Dienst-Nutzer, *Shared UIs*, Dienst-Ökosytem und Dienst-Anbietern dar. Die Abbildung konkretisiert das Prinzip am Beispiel aus Abschnitt 2.5.1. Ein Dienst-Nutzer wählt aus einer Menge frei verfügbarer *Shared UIs* die für seinen Anwendungsfall passenden Komponenten aus und bindet diese als Komposition in den Fragefluss einer Rahmenanwendung ein ①. In der Abbildung werden *Shared UIs* für die *Flug- und Hotelbuchung* sowie die *Reiseversicherung* eingebunden.

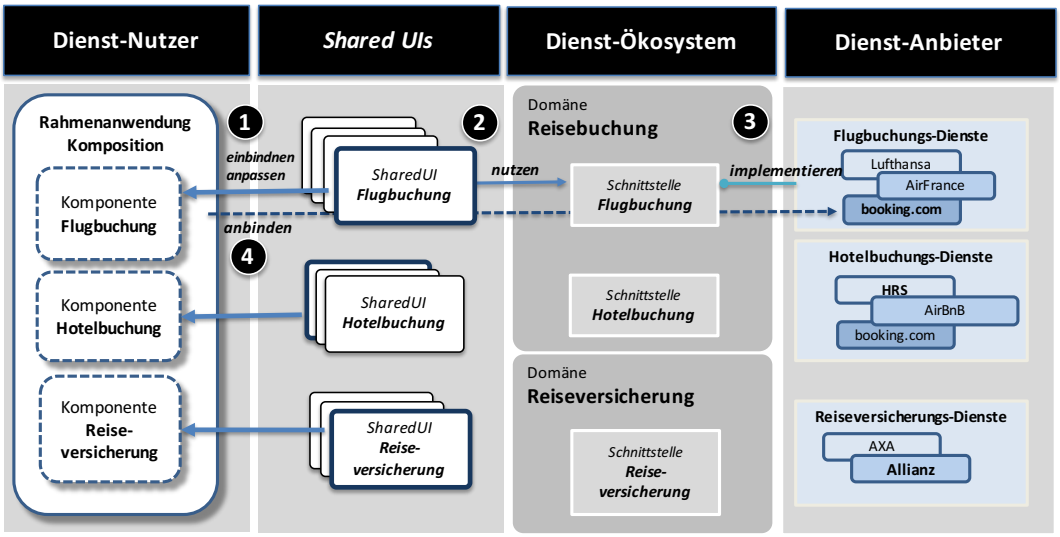


Abbildung 2.10. Prinzip von *Shared UIs*

Die *Shared UIs* wurden für die Erfassung der Daten spezifischer Dienste eines Dienst-Ökosystems erstellt ②. Sie basieren auf deren Schnittstellenbeschreibungen und erfassen die Daten in einer Form, die zur Kommunikation mit konkreten Dienst-Implementierungen geeignet ist ③. Zur Laufzeit bindet die Rahmenanwendung konkrete Dienst-Anbieter an ④, an welche die erfassten Daten der Komponenten übermittelt werden. In der Abbildung werden z.B. zur Durchführung der *Flugbuchung* die zur Flugbuchungs-Schnittstelle passenden Daten zur Reservierung eines Fluges an einen Dienst des Anbieters *booking.com* gesendet.

Um in Kompositionen nutzbar zu sein, müssen *Shared UIs* als Komponenten eingebunden werden können. Sie weisen die in Abschnitt 2.5.2 genannten Komponenteneigenschaften auf. Hinzu kommen Eigenschaften, die deren gemeinschaftliche Nutzbarkeit ermöglichen:

- Erstellbarkeit durch beliebige Anbieter
- Beschreibung in einer standardisierten, verteilbaren Form
- Erfassung der Eingabedaten für Dienste eines Ökosystems

- Nutzbarkeit der erfassten Daten mit beliebigen Dienst-Anbietern

Shared UIs zeichnen sich dadurch aus, dass sie in einer universellen Form beschrieben sind, die von beliebigen Anbietern zur Verfügung gestellt und von beliebigen Nutzern interpretiert werden kann. Sie erfassen die Eingabedaten für konkrete Dienste eines Ökosystems und stellen diese in einer Form zur Verfügung, die zur Kommunikation mit verschiedenen Dienst-Anbietern nutzbar ist.

2.6 Modellgetriebene Entwicklung von Benutzungsschnittstellen

Modelle bilden die zentrale Basis bei der Entwicklung von IT-Anwendungen und sind ein fundamentales Konzept des Software-Engineerings [134]. Insbesondere werden formalisierte Modelle vermehrt zur Generierung von Artefakten eines Softwaresystems bis hin zu vollständig lauffähigen Anwendungen eingesetzt und damit die Automatisierung der Software-Herstellung erhöht [169]. In diesem Zuge entstanden Methoden zur **modellgetriebenen Entwicklung** (Model-Driven Engineering - **MDE**)[22], in denen formale Modelle die Grundlage des Entwicklungsprozesses bilden [84, 142, 215].

Das Versprechen der modellgetriebenen Entwicklung liegt in der Flexibilisierung der Software-Erstellung [144], die sich insbesondere bei Anwendungen mit langer Lebensdauer auszahlt. Durch die maschinelle Erstellung kann z.B. auf neue Technologieentwicklungen und wechselnde Anforderungen bei der Integration von Anwendungen in angeschlossene Systemen reagiert werden. Dies verringert Wartungsaufwände und führt zu einer höheren Softwarequalität bei der Weiterentwicklung und Migration von Anwendungen [215].

In den folgenden Abschnitten werden die Grundzüge der modellgetriebenen Entwicklung und deren Anwendung auf die automatische Erstellung von Benutzungsschnittstellen dargestellt.

2.6.1 Konzept der modellgetriebenen Entwicklung

Mellor et al. [141] fasst die Grundidee der modellgetriebenen Entwicklung vereinfacht zusammen:

> "Model-driven development is […] the notion that we can construct a model of a system that we can then transform into the real thing"

Sie liegt darin, Anwendungssysteme in einer abstrahierten, maschinell verarbeitbaren Form zu beschreiben und diese in konkrete Implementierungen zu transformieren. Softwarekomponenten werden unabhängig von ihrer technischen Umsetzung in einem abstrakten Modell spezifiziert. Dieses wird in Schritten sukzessive konkretisiert und in ein **plattformspezifisches[23] Modell** transformiert. Der Übergang von abstrakteren hin zu plattformspezifischen Modellen erfolgt dabei **automatisiert** durch die Verwendung von Transformationswerkzeugen.

[22] Als synonyme Bezeichnungen finden sich in der Literatur Model-Driven Software Development (MDSD) sowie Model-Driven Development (MDD).

[23] Der Begriff der *Plattform* muss dabei abstrahiert aufgefasst werden. Hierunter ist die Zielumgebung / das Umfeld zu verstehen, für welche ein abstraktes Modell konkretisiert wird. Es handelt sich dabei nicht notwendigerweise um eine Ablaufumgebung.

Abbildung 2.11 zeigt die Vorgehensweise am Beispiel des MDD-Ansatzes, der von der *Object Management Group* [144] vorgeschlagen wird. Der Ansatz unterscheidet zwischen einem *plattformunabhängigen Model* (Platform Independent Model - **PIM**) und einem *plattformspezifischen Modell* (Platform Specific Model - **PSM**) als konzeptionelle Basis.

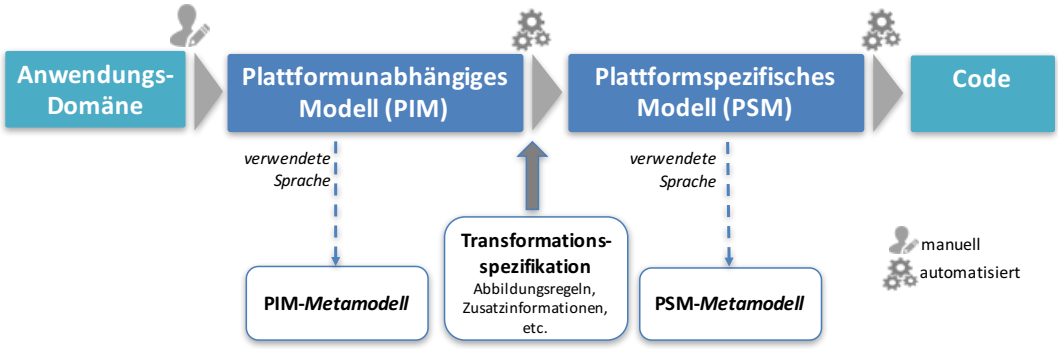


Abbildung 2.11. Transformation als Grundprinzip der MDE

Im ersten Schritt wird (meist manuell) ein für eine Anwendungsdomäne passendes PIM der Anwendung erstellt. Das PIM wird in einer plattformunabhängigen Sprache entwickelt, die durch ein entsprechendes Metamodell spezifiziert ist (*PIM-Metamodel*). Im nächsten Schritt wird eine Zielplattform gewählt, für die ein PSM erzeugt werden soll. Die Sprache der Zielplattform ist durch ein *PSM-Metamodel* spezifiziert. Eine Transformationsspezifikation beschreibt die Abbildung der Elemente der Metamodelle aufeinander und bestimmt die Transformation des PIM in das PSM [144]. Aus dem PSM kann abschließend der konkrete Code für Artefakte auf der Zielplattform generiert werden.

Die derzeit existierenden MDE-Methoden, deren **Modelle** und **Notationen** sind heterogen. Sie unterscheiden sich insbesondere im Abstraktionsgrad der Modellierung. Der *Model Driven Architecture*-Ansatz der OMG [144] schlägt beispielsweise einen Prozess für die Modellierung und Transformationen vor, der standardisierte Modelle als Grundlage verwendet. Der Ansatz ist generisch auf eine Vielzahl von Problemstellungen anwendbar.

Zur Vereinfachung der Modellerstellung, werden vermehrt **domänenspezifische Sprachen** (Domain-Specific Language – DSL) eingesetzt [71, 215, 230]. Sie bieten auf konkrete Anwendungsfelder zugeschnittene Modellierungssprachen mit dem Ziel, den Modellierungsaufwand zu minimieren. Diese werden analog dem dargestellten MDE-Grundprinzip der OMG in konkrete Artefakte transformiert. Die Werkzeuge zur Transformation der DSL auf die Zielplattform sind nicht festgelegt. Häufig werden hierfür Werkzeuge zur Erstellung von Compilern (z.B. *ANTLR*[24]) oder Produkte zur Entwicklung und Verarbeitung von DSLs (z.B. *Xtext*[25]) verwendet.

2.6.2 Anwendung auf Benutzungsschnittstellen

Die Forschungen zur automatischen Generierung von Benutzungsschnittstellen (*Model Driven UI Development* - MDUID) reichen in die 1980er Jahre zurück. Seit dieser Zeit sind eine Vielzahl von Ansätzen entwickelt worden, die aus abstrakten Modellen lauffähige und technologiespezifische Benutzungsschnittstellen erzeugen [85, 140].

[24] http://www.antlr.org/

[25] http://www.eclipse.org/Xtext/

Das **Ziel der MDUID-Ansätze** liegt darin, eine Abstraktion und Modellierung für Benutzungsschnittstellen zu finden, welche mit modellgetriebenen Methoden in ausführbare Artefakte transformiert werden kann. Die Herausforderungen bei der Modellierung liegen auf zwei Ebenen. Einerseits müssen die Modelle hinreichend Informationen zur Generierung hochwertiger Benutzungsschnittstellen enthalten. Andererseits müssen sie die Diversität der Zielplattformen unterstützen.

Die **Herausforderungen** liegen darin, eine Sprache zu finden, welche Interaktionen, Logik und Präsentationsformen einer Anwendung hinreichend abstrahiert beschreibt, dass UIs für beliebige Plattformen generisch erstellt werden können. Die Modelle und Verfahren müssen folgende Aspekte unterstützen [140]:

- Diversität der Benutzer mit unterschiedliche Anforderungen bzgl. Präferenzen, Fähigkeiten, Sprache und Erfahrungen
- Unterschiedliche Eingabemöglichkeiten und Modalitäten der Zielplattformen
- Heterogene Zielplattformen: Programmier- und Markupsprachen sowie Werkzeuge
- Heterogenität der Arbeitsumgebungen

In Bezug auf die Modellierung und des Vorgehens entwickelte sich in den vergangenen Jahre ein weitgehend akzeptiertes Verständnis darüber, welche Abstraktionen und Modelltypen sinnvoll sind. Bezüglich der in den einzelnen Modellen enthaltenen Semantik und Information besteht trotz der langjährigen Forschung hingegen kein Konsens [140].

Zur Erfassung der konzeptionell relevanten Modelle wurden verschiedene Frameworks vorgeschlagen. In Silva [211] ist eine erste Architektur für die modellgetriebene Entwicklung von Benutzungsschnittstellen beschrieben, die auf Task- und Dialogmodellen basiert. Calvary et al. [26] formalisieren diesen Ansatz und schlagen ein Referenzframework vor, welches in der Literatur bis heute häufig als Basis verwendet wird (s. Abschnitt 2.6.3).

Diese Ansätze identifizieren drei Arten von Modellen, die zur Generierung einer UI sinnvoll sind: Tasks & Konzeptmodell, Dialogmodell und Präsentationsmodell [140]. Das **Task- & Konzeptmodell** beschreibt die Aufgaben und verarbeiteten Daten, die vom interaktiven System in seiner Gesamtheit bearbeitet werden können und deren Beziehung zueinander. Das **Dialogmodell** wird aus dem Taskmodell abgeleitet und beschreibt die Aktivitäten und Transitionen, die innerhalb einer Anwendung auftreten können. Das **Präsentationsmodell** repräsentiert die darzustellende Benutzungsschnittstelle für das Dialogmodell (z.B. grafisch, sprachbasiert).

2.6.3 Das CAMELEON-Referenzframework

Das CAMELEON-Referenzframework (CRF) [25] ist ein konzeptioneller und methodischer Rahmen zur systematischen Generierung von Benutzungsschnittstellen [223]. Das Ziel ist die Schaffung eines Vorgehensmodells für den Entwurf und die Entwicklung von qualitativ hochwertigen, multi-modalen, kontextsensitiven und interaktiven Benutzungsschnittstellen[26]. In Abbildung 2.12 sind die Elemente des Referenzframeworks dargestellt.

Das CRF unterscheidet vier Abstraktionsebenen, in welchen Modelle für Benutzungsschnittstellen unterschieden werden. Die abstrakteste Sicht ist die **Concepts- & Tasks (CT)**-Ebene.

[26] vgl. http://giove.isti.cnr.it/projects/cameleon/goals.html

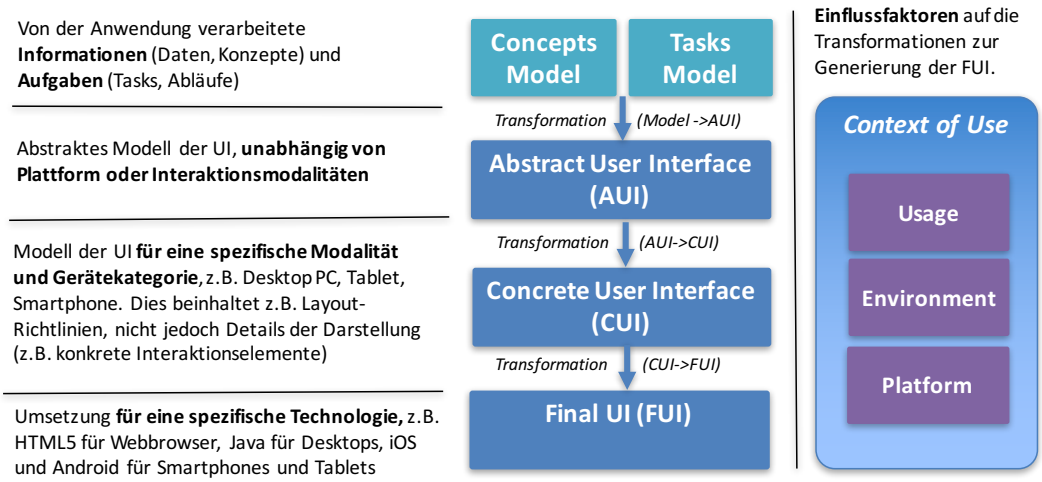


Abbildung 2.12. Elemente des CAMELEON-Referenzframeworks

Sie beschreibt Aufgaben, deren zeitliche Abfolge (Dialogabläufe) und die Daten, die von der Benutzungsschnittstelle während der Interaktion verarbeitet werden sollen.

Auf der **Abstract UI (AUI)**-Ebene werden Benutzungsschnittstellen abstrakt, d.h. unabhängig von einer Plattform oder Modalität beschrieben. Sie beschreiben die logische Struktur der UI (z.B. temporale Abfolge, Gruppierung von Daten) und abstrahierte Interaktiosobjekte [228]. AUI-Modelle eignen sich damit als Grundlage zur Erzeugung von UIs für unterschiedliche Interaktions- und Technologiekontexte.

Auf der **Concrete UI (CUI)**-Ebene sind Benutzungsschnittstellen für konkrete Modalitäten und Gerätekategorien beschrieben. CUI-Modelle konkretisieren diese hinsichtlich der Struktur (z.B. Verteilung auf Darstellungseinheiten) und der Interaktionsobjekte. Sie beschreiben den Aufbau der UI, unabhängig von einer konkreten Zieltechnologie.

Die **Final UI (FUI)**-Ebene beschreibt die Benutzungsschnittstelle für eine konkrete Zielplattform. FUI-Modelle bestehen aus technologiespezifischen Artefakten, welche auf der Zielplattform interpretiert werden können (z.B. Java- oder HTML-Code).

Die einzelnen Ebenen stellen eine zunehmende Konkretisierung der Modelle hin zu einem finalen Modell dar. Das CRF schlägt einen Prozess vor, der Modelle einer abstrakten Stufe durch Transformation in ein FUI-Modell überführt[27]. Der Verfeinerungsprozess (*reification*) besteht aus vier Transformationsschritten. Hierbei wird ein CT-Model sukzessive in ein AUI- und CUI-Modell transformiert und abschließend ein FUI-Modell der Zielplattform abgeleitet. Die Transformationen zwischen den Ebenen werden dabei durch Einflussfaktoren wie die Nutzung, Umgebung und Plattform (*Context-of-Use*) bestimmt. Dieser enthält Informationen, die von den Transformationen zur Konkretisierung ausgewertet werden.

Das CRF legt weder die Inhalte der Modelle einer Abstraktionsstufe noch die Informationen des *Context-of-Use* fest. Dies bleibt konkreten Umsetzungen des Ansatzes überlassen.

[27] Das CRF sieht zusätzlich Transformationen vor, die ein konkretes in ein abstrakteres Modell (*abstraction*) bzw. in ein Modell der gleichen Abstraktionsebene überführen (*translation*). Dieser Aspekt wird hier nicht näher betrachtet und auf [25] verwiesen.

3. Anforderungen, Stand der Forschung und Lösungsansatz

Das Ziel dieses Kapitels ist es, eine Übersicht über den in der Arbeit verfolgten Lösungsansatz für die in Abschnitt 1.3 dargestellten Herausforderungen zu geben. Hierzu werden zuerst die Anforderungen formuliert, die sich aus den in Abschnitt 1.3 genannten Problemstellungen und den Grundlagen aus Kapitel 2 ergeben. Anschließend wird der Stand der Forschung dargestellt und die Lücke aufgezeigt, die bei den bisherigen Ansätzen bezüglich der Anforderungen besteht.

3.1 Anforderungen an die Lösung

Durch die Verwendung eines modellgetriebenen Ansatzes sollen folgende Verbesserungen gegenüber bestehenden Ansätzen erreicht werden (vgl. Abschnitt 1.3):

- Vereinfachung der Erstellung und Wartung von Benutzungsschnittstellen
- Geringere Komplexität und Redundanz bei der Modellierung von Varianten
- Automatische Erzeugung nicht-trivialer Benutzungsschnittstellen
- Erhöhte Wiederverwendbarkeit der Modelle in variierenden Szenarien
- Gemeinschaftliche Nutzbarkeit der Modelle in Dienst-Ökosystemen

In Abschnitt 1.2 wurden die Herausforderungen aufgezeigt, die im Rahmen der Arbeit zu lösen sind. Zudem wurden die Eigenschaften dargestellt, die ein modellgetriebener Ansatz hierzu aufweisen muss. In Kapitel 2 wurden die damit verbundenen Aufgabenstellungen in den Abschnitten 2.2-2.5 detailliert. Aus diesen Darstellungen ergeben sich zusammengefasst folgende Aufgabenstellungen, die im Lösungsansatz umzusetzen sind:

- (a) Modellierung automatisch generierbarer Benutzungschnittstellen
- (b) Modellierung inhaltlicher Varianten für unterschiedliche Nutzergruppen
- (c) Komposition und Anpassung bestehender Modelle in neuen Umgebungen
- (d) Generierung technischer Varianten für multiple Plattformen und Technologien
- (e) Gemeinschaftliche Nutzung bestehender Modelle für Dienste eines Ökosystems
- (f) Verwendbarkeit der generierten Ergebnisse mit beliebigen Dienst-Anbietern

Entsprechend der Beiträge der Arbeit aus Abschnitt 1.4 werden die Anforderungen in folgenden Bereichen dargestellt:

- **(A.1)** Modellierung von Benutzungsschnittstellen und Varianten
- **(A.2)** Komposition von Benutzungsschnittstellen
- **(A.3)** Verfahren zur Generierung von Varianten
- **(A.4)** Wiederverwendung als *Shared UIs* in einem Dienst-Ökosystem

Für jeden Bereich werden die Anforderungen aufgeführt und dargestellt, inwiefern hierdurch die o.g. Aufgabenstellungen adressiert werden. Die Anforderungen werden abschließend in Tabelle 3.1 zusammengefasst.

(A.1) Modellierung von Benutzungsschnittstellen und Varianten

Benutzungsschnittstellen besitzen einen strukturierten Aufbau und weisen ein dynamisches Verhalten auf (vgl. Abschnitt 2.2). Die Inhalte, das Verhalten sowie das Aussehen variieren dabei anhängig vom Nutzungs-, Interaktions- und Technologiekontext (vgl. Abschnitt 2.3).

In den Abschnitten 2.2.2 und 2.3.2 sind die Eigenschaften dialogbasierter Anwendungen und deren Varianten beschrieben. Sollen aus einem Modell vollständig automatisiert nutzergerechte Benutzungsschnittstellen mit diesen Eigenschaften hergeleitet werden, stellt dies folgende Anforderungen an die Modellierung:

- **(A1.1): Vollständigkeit des Modells**
- **(A1.2): Eignung zur Erzeugung technischer Varianten**
- **(A1.3): Eignung zur redundanzfreien Beschreibung inhaltlicher Varianten**

Zu (A1.1): Das Modell muss Struktur und Verhalten der Benutzungsschnittstelle **im Sinne der Generierbarkeit vollständig** beschreiben. D.h. es muss alle zur automatisierten Herleitung relevanten Informationen enthalten. Nach der Definition von Chlebek (vgl. Abschnitt 2.2.2) sind dies Informationen zu *Struktur und Aufbau*, *Interaktionselementen zur Ein-/Ausgabe* der Daten und *Verhalten* der Benutzungsschnittstelle. Diese Aspekte sind elementar und ermöglichen nach Chlebek eine vollständige Beschreibung.

Zu (A1.2): Um aus dem Modell automatisiert **technische Varianten** erzeugen zu können, muss das Modell unabhängig vom konkreten Interaktions- und Technologiekontext sein und dennoch alle Informationen enthalten, die zur Generierung technologiespezifischer Eigenschaften (Navigation, Darstellungseinheiten, Interaktionselemente) relevant sind (vgl. Abschnitt 2.3). Daher wird ein **technologieneutrales Modell** benötigt, aus dem verschiedene, nicht-triviale technische Varianten herleitbar sind.

Zu (A1.3): Inhaltliche Varianten unterscheiden sich abhängig vom Nutzerkontext im Umfang der erfassten Daten und weisen ggf. ein angepasstes Verhalten auf (vgl. Abschnitt 2.3). Zu deren Modellierung müssen diese Unterschiede beschrieben werden können. Um dem Ziel der vereinfachten Erstellung und Wartung gerecht zu werden, soll die Modellierung differentiell erfolgen können und Unterschiede zum Ausgangsmodell modelliert werden.

Die Erfüllung von **(A1.1)** ermöglicht die vollständige Generierbarkeit von Benutzungsschnittstellen und ist die Grundvoraussetzung zu Erreichung der Ziele der Arbeit. Die Anforderung **(A1.2)** erweitert den Modellfokus auf die Generierbarkeit spezifischer Varianten für multiple technische Kontexte. Die Anforderung **(A1.3)** fügt die Erstellung nutzergruppenspezifischer Varianten hinzu.

*Durch die Anforderungen werden die o.g. Aufgabenstellungen **(a)** und **(b)** abgedeckt. Zudem bilden sie die Grundlage für die unter **(d)** aufgeführte Aufgabenstellung.*

(A.2) Komposition von Benutzungsschnittstellen

Um existierende Modelle zu funktional höherwertigen Anwendungen kombinieren zu können, müssen sie in den Dialogablauf einer Rahmenanwendung integriert werden können (vgl.

Abschnitt 2.4). In Abschnitt 2.5.2 werden die Eigenschaften von Kompositionen dargestellt, aus welchen sich folgende Anforderungen an die Modellierung ergeben:

- **(A2.1): Freie Kombinierbarkeit von Modellen**
- **(A2.2): Anpassbarkeit der Komponenten an ihr Umfeld**
- **(A2.3): Vollständigkeit der Komposition**

Zu (A2.1): Bestehende Benutzungsschnittstellen müssen als Komponenten an beliebigen Stellen einer Anwendung eingefügt werden können. Sie müssen dabei ihre Funktionalität erhalten und als Teil der umgebenden Anwendung erscheinen (*seamless integration*).

Zu (A2.2): Um Anwendungen flexibel in ein neues Umfeld zu integrieren, müssen sowohl Inhalt als auch Verhalten einer Komponente der neuen **Nutzung angepasst** werden können.

Zu (A2.3): Die Komposition und ihre Komponenten müssen dabei stets **vollständig** im Sinne der Generierbarkeit sein, damit die Erzeugung der Benutzungsschnittstelle der Komposition automatisiert möglich ist. Zusätzlich wird gefordert, dass die Komposition wiederum zur Erzeugung inhaltlicher und technischer Varianten geeignet ist. Sie müssen daher die bereits unter (A.1) gestellten Anforderungen erfüllen.

Die Erfüllung von **(A2.1)** ermöglicht die Kombination bestehender Benutzungsschnittstellen zu neuen Anwendungen. Die Anforderung **(A2.2)** ermöglicht dabei deren nachträgliche Anpassung an ein neues Umfeld und erhöht dadurch die Wiederverwendbarkeit von Komponenten. Anforderung **(A3.3)** garantiert zusätzlich, dass Kompositionen als Ganzes weiterhin vollständig automatisiert erzeugt werden können.

*Durch die Anforderungen wird die unter **(c)** genannte Aufgabenstellung abgedeckt.*

(A.3) Verfahren zur Generierung von Varianten

Aus den Modellen müssen automatisiert funktionsfähige Benutzungsschnittstellen für multiple Modalitäten erzeugt werden können (vgl. Abschnitt 2.3). Es wird ein Verfahren benötigt, das aus den in (A.1) und (A.2) geforderten Modellen automatisiert ausführbare Schnittstellen für multiple Modalitäten, Interaktions- und Technologiekontexte erzeugt. Das Verfahren muss folgende Anforderungen erfüllen:

- **(A3.1): Herleitung inhaltlicher Varianten und Kompositionen**
- **(A3.2): Generierung technischer Varianten**
- **(A3.3): Ausführbarkeit in einer Laufzeitumgebung**

Zu (A3.1): Das Verfahren muss aus den in (A.1) und (A.2) geforderten Modellen automatisiert eine inhaltliche Variante erzeugen können, indem es Modelle kombiniert und inhaltlich einem vorgegebenen Nutzerkontext anpasst.

Zu (A3.2): Das Verfahren muss vollständig automatisiert **technische Varianten** für unterschiedliche Interaktions- und Technologiekontexte generieren (z.B. grafische oder sprachbasierte Benutzungsschnittstellen). Abhängig vom Interaktions- und Technologiekontext müssen dabei geeignete Interaktionselemente gewählt und auf die verfügbaren Technologien der Zielplattformen abgebildet werden.

Zu (A3.3): Das Resultat muss eine finale Benutzungsschnittstelle darstellen, die in einer **konkreten Laufzeitumgebung ausgeführt werden** kann. Es muss eine plattformspezifische Umsetzung liefern und dabei das im Modell beschriebene Verhalten abbilden.

Die Erfüllung von **(A3.1)** ermöglicht die weiterhin vollständig automatisierte Erzeugung von Varianten und Kompositionen. **(A3.2)** und **(A3.3)** fordern darüber hinaus die Generierung lauffähiger, auf spezifische technische Kontexte zugeschnittener Anwendungen.

*Hierdurch wird die unter **(d)** genannte Aufgabenstellung abgedeckt.*

(A.4) Wiederverwendung als *Shared UIs* in einem Dienst-Ökosystem

Shared UIs stellen selbständig funktionsfähige Komponenten dar, die in Anwendungen an eine spezifische Nutzung angepasst integriert und in einem spezifischen Interaktions- und Technologiekontext ausgeführt werden können (vgl. Abschnitt 2.5). Grundsätzlich gelten für die Modellierung von *Shared UIs* **alle Anforderungen, die bereits unter (A.1) und (A.2) aufgeführt** wurden.

In Abschnitt 2.5.2 sind die Eigenschaften von *Shared UIs* aufgeführt. Ihre gemeinschaftliche Nutzung wird dadurch ermöglicht, dass sie in einer standardisierbaren, verteilbaren Form vorliegen und von beliebigen Anbietern der Allgemeinheit zur Verfügung gestellt werden können. Sie sind für eine spezifische Dienst-Schnittstelle erstellt und können mit entsprechenden Dienst-Implementierungen in einem Ökosystem zusammenarbeiten. Das Prinzip der *Shared UIs* ermöglicht dem Nutzer potentiell, frei zu entscheiden, welche Benutzungsschnittstelle er aus welcher Quelle bezieht, welche konkrete Variante erzeugt werden und welcher Dienst-Anbieter erfasste Daten verarbeiten soll. Daraus ergeben sich ergänzende Anforderungen an den Lösungsansatz:

- **(A4.1):** Verwendung einer universell interpretierbaren Repräsentation
- **(A4.2):** Bereitstellung erfasster Daten für Dienste im Service-Ökosystem
- **(A4.3):** Verteil- und Verwendbarkeit in dezentraler Infrastruktur

Zu (A4.1): Um von anderen Anwendungen verarbeitet werden zu können, muss das Modell in einer standardisierten und damit für potentielle Nutzer interpretierbaren Form vorliegen. Diese muss unabhängig vom technischen und fachlichen Umfeld der jeweiligen Anwendung sein (vgl. Abschnitt 2.5.2). Es wird eine universelle Repräsentation gefordert, welche alle unter (A.1) und (A.2) genannten Anforderungen erfüllt.

Zu (A4.2): Um die Anbindung an Dienst-Implementierungen automatisieren zu können, müssen die Komponenten erfasste Daten entsprechend der Dienst-Schnittstellen bereitstellen. Im Modell müssen daher alle Informationen enthalten sein, um aus erfassten Daten eine für Dienste des Ökosystems nutzbare Repräsentation zu erzeugen.

Zu (A4.3): Um *Shared UIs* gemeinschaftlich nutzbar zu machen, müssen die Modelle in einer dezentralen Infrastruktur verwendbar sein. Die Komponenten müssen einerseits eine vollständige Benutzungsschnittstelle beschreiben, andererseits für den Nutzer mit geringem Aufwand zu beschaffen sein. Dies erfordert, dass die Repräsentation als vollständig funktionale Einheit verteilt werden kann.

Die Erfüllung von **(A4.1)** sichert die Nutzbarkeit und automatische Generierbarkeit von Benutzungsschnittstellen durch Dritte. Beliebige Anbieter können für beliebige Nutzer voll-

ständig generierbare UIs, Varianten und Komponenten bereitstellen. Die Anforderung **(A4.2)** ermöglicht darüber hinaus deren Nutzung mit multiplen Dienst-Implementierungen. **(A4.3)** schafft ergänzend die Voraussetzungen zur gemeinschaftlichen Nutzung und Verteilbarkeit.

*Hierdurch werden die unter **(e)** und **(f)** genannten Aufgabenstellungen abgedeckt.*

Zusammenfassung der Anforderungen. Tabelle 3.1 fasst die Anforderungen in den Bereichen (A.1) bis (A.4) zusammen. Sie dienen im Folgenden zur Bewertung bestehender Ansätze, der Identifikation der Forschungslücke sowie zur Umsetzung der Beiträge der Arbeit.

	Anforderung	Erläuterung
A.1	**Modellierung von Benutzungsschnittstellen und Varianten**	
(A1.1)	**Vollständigkeit des Modells**	Ein Modell als Grundlage, aus dem **qualitativ hochwertige** Benutzungsschnittstelolen hinsichtlich Struktur, Interaktionselementen und Verhalten **vollständig generiert** werden können.
(A1.2)	**Eignung zur Erzeugung technischer Varianten**	Ein **technologieneutrales** Modell, das von konkreten Nutzungs-, Interaktions- und Technologiekontexten abstrahiert und aus dem dennoch nicht-triviale Interaktionselemente unterschiedlicher Technologien hergeleitet werden können
(A1.3)	**Eignung zur redundanzfreien Beschreibung inhaltlicher Varianten**	Ein Modell, das die Möglichkeit zur **differentiellen Spezifikation** inhaltlicher Varianten ermöglicht
A.2	**Komposition von Benutzungsschnittstellen**	
(A2.1)	**Freie Kombinierbarkeit von Modellen**	Möglichkeit zur Integration von existierenden Modellen als Komponenten an **beliebigen Stellen** in Dialogabläufen von Kompositionen
(A2.2)	**Anpassbarkeit der Komponenten an ihr Umfeld**	Anpassbarkeit von **Inhalt und Verhalten** von Komponenten an die konkrete Nutzung einer Komposition
(A2.3)	**Vollständigkeit der Komposition**	▪ **Vollständige Generierbarkeit** für den technischen Kontext der Komposition gemäß (A1.1) ▪ Verwendbarkeit von Kompositionen in **inhaltlichen und technischen Varianten, sowie in anderen Kompositionen**
A.3	**Verfahren zur Generierung von Varianten**	
(A3.1)	**Herleitung inhaltlicher Varianten und Kompositionen**	▪ **Herleitung inhaltlicher Varianten** durch Anapssung des Inhalts und des Verhaltens ▪ **Herleitung von Kompositionen** durch Integration von Komponenten in den Dialogfluss und Anpassung deren Inhalts und Verhaltens
(A3.2)	**Generierung technischer Varianten**	Generierung finaler Benutzungsschnittstellen aus dem Modell für **beliebige Interaktions- und Technologiekontexte**
(A3.3)	**Ausführbarkeit in Laufzeitumgebung**	Vollständige Generierung auf einer **spezifischen Zielplattform ausführbarer** Anwendungen
A.4	**Wiederverwendung in einem Dienst-Ökosystem**	
(A4.1)	**Verwendung einer universell interpretierbaren Repräsentation**	▪ Beschreibung des Modells in einer nicht-proprietären Form, die in unterschiedlichen Nutzungsumgebungen interpretiert und verarbeitet werden kann ▪ Erfüllung der Anforderungen (A.1) und (A.2) und Eignung zur vollständigen Generierung gemäß (A.3)
(A4.2)	**Bereitstellung erfasster Daten für Dienste im Service-Ökosystem**	▪ Modellierung aller Informationen, die zur **Anbindung an konkrete Dienste erforderlich** sind ▪ Möglichkeit zur Erzeugung der Eingabedaten in einer für Dienst-Schnittstellen verwendbaren Form
(A4.3)	**Verteil- und Verwendbarkeit in einer dezentralen Infrastruktur**	▪ **Verteilbarkeit** des Modells über eine geringe Zahl von Artefakten, die dezentral zur Verfügung gestellt werden können ▪ **Unabhängigkeit** des Modells von **spezifischen Generatoren und** konkreten **Dienstanbietern**

Tabelle 3.1. Anforderungen an die Lösung

3.2 Stand der Forschung

Dieser Abschnitt stellt bestehende Ansätze zur automatischen Erzeugung von Benutzungsschnittstellen dar, bewertet diese hinsichtlich der in Abschnitt 3.1 postulierten Anforderungen und identifiziert ungelöste Problemstellungen, die in der Arbeit gelöst werden.

3.2.1 Historische Entwicklung *Model Driven UI Development*

In Meixner et al. [140] wird die historische Entwicklung modellgetriebener Ansätze zur UI-Generierung in Generationen eingeteilt, die sich im Umfang der unterstützten Aspekte unterscheiden.

Die Ansätze der **ersten Generation** (ca. 1990-1996) fokussierten auf die Abstraktion und Beschreibung relevanter Aspekte von (grafischen) Benutzungsschnittstellen, die zur automatischen Generierung erforderlich sind. Hauptsächlich entstanden hierbei abstrakte Modelle zur deklarativen Beschreibung der Präsentation von Benutzungsschnittstellen. Aus diesen konnten Artefakten generiert werden, die (meist manuell) in eine lauffähige Anwendungen integriert wurden. Beispiele sind UIDE [68] und HUMANOID [219]. In der **zweiten Generation** (ca. 1995-2000) fand eine Diversifizierung der Modellierung statt: einzelne Aspekte der Benutzungsschnittstellen wurden in Teilmodellen beschrieben, die zusätzliche Informationen zum Ablauf und der Ausführung der Anwendung enthalten. Insbesondere wurde die Taskmodellierung (z.B. *Concurrent Task Trees*) integraler Bestandteil der Modellierung. Beispiele sind ADEPT [45], TRIDENT [227] und MASTERMIND [220].

Die wachsende Zahl unterschiedlicher Plattformen und Geräte (z.B. mobile Geräte wie Smartphones, PDAs) führte zu einer **dritten Generation** von Ansätzen (ca. 2000–2004), die sich auf die geräteunabhängige Beschreibung von Benutzungsschnittstellen konzentrierten. Beispiele sind TERESA [148] und Dygimes [40]. Die **vierte Generation** (ab ca. 2004) erweitert diese um Ansätze zur Modellierung von Anwendungen, die für eine Vielzahl von Plattformen, Geräten und Modalitäten verwendet werden können. Weitere Entwicklungen befassen sich mit der Optimierung der Nutzbarkeit (Usability) automatisch erzeugter Benutzungsschnittstellen.

3.2.2 Kategorisierung und Bewertung bestehender Ansätze

Die in den vergangenen Jahren entwickelten Ansätze zur modellgetriebenen Entwicklung von Benutzungsschnittstellen weisen untereinander Ähnlichkeiten auf und können anhand des CAMELEON-Referenzframeworks (**CRF**, vgl. Abschnitt 2.6.3) kategorisiert werden. Anhand der Abstraktionsstufen der manuell zu erstellenden Modelle, deren Zahl und Inhalte kann bewertet werden, inwieweit die zugeordneten Ansätze einer Kategorie zur vollständigen Automatisierung unter den gestellten Anforderungen geeignet sind.

Abbildung 3.1 zeigt im Zentrum die Abstraktionsstufen und Modelltypen des CRF. Daneben sind die Kategorien aufgeführt, die zur Einordnung bestehender Ansätze in dieser Arbeit verwendet werden. Zu jeder Kategorie ist eine kurze Beschreibung des Modellierungsansatzes sowie die Abstraktionsstufen angegeben, auf denen Modelle manuell erstellt werden.

Im Folgenden werden die Kategorien beschrieben, deren Eigenschaften und konkrete Ansätze dargestellt. Zu jeder Kategorie wird abschließend eine **Bewertung** angegeben, die aufzeigt, **inwieweit die Anforderungen der Arbeit durch die enthaltenen Ansätze abgedeckt**

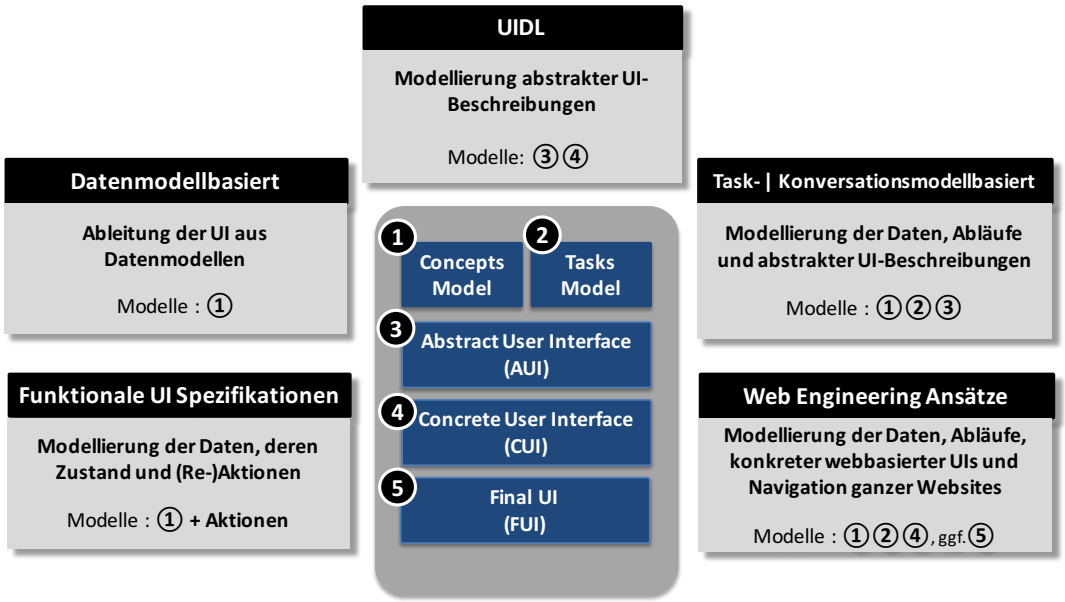


Abbildung 3.1. Kategorisierung bestehender Ansätze über das CAMELEON-Referenzframework

werden. Dies dient als Grundlage zur Identifikation der ungelösten Problemstellungen (Forschungslücke) in Abschnitt 3.2.3.

User Interface Description Languages (UIDL)

Der Ansatz von UIDL besteht in der Verwendung abstrahierter Modelle einer UI, um daraus plattformspezifische Varianten zu erzeugen. In bestehenden Ansätzen werden hierzu Modelle auf der AUI- bzw. CUI-Ebene erstellt, welche Struktur, Eingabeelemente und Verhalten beschreiben. Während sich frühe Ansätze auf die Beschreibung und Generierung von Eingabeelementen für spezifische Plattformen (z.B. webbasierte Anwendungen, Desktop-Anwendungen) beschränkten, kamen später weitere Aspekte hinzu, z.B. Elementtypen, Validierungen und die Beschreibung des Verhaltens. Die Beschreibungen wurden generischer und konnten so für verschiedene Modalitäten eingesetzt werden.

Beispiele für UIDLs auf der CUI-Ebene sind neben **HTML** die *Extensible Application Markup Language* **XAML** [29], die *JavaFX Markup Language* **FXML** [64] und *Layout Resource*-Dateien[30], welche zur Beschreibung von UIs für spezifische Plattformen (Windows, Java, Android) genutzt werden.

XUL [149] ist hingegen eine Lösung zur plattformunabhängigen Beschreibung grafischer Benutzungsschnittstellen. XUL modelliert abstrakte Interaktionselemente und fügt Layout-Informationen für formularbasierte Anwendungen hinzu. Benutzungsschnittstellen werden über Layout- (z.B. *Windows*, *Frames*, *Boxes*, *Layouts*), Interaktionselemente (*XUL Widgets*) und Stilangaben spezifiziert. Programmlogik zur Umsetzung des Verhaltens kann über sog. *Bindings* Interaktionselementen hinzugefügt werden. Die in XUL vorgesehenen Elemente beschränken den Einsatz jedoch auf grafische UIs. Zur Beschreibung werden sowohl CUI- als auch AUI-Modelle verwendet.

[29] https://msdn.microsoft.com/de-de/library/cc295302.aspx

[30] https://developer.android.com/guide/topics/resources/layout-resource.html

XForms [53] wurde vom World Wide Web Consortium (W3C) als Ansatz zur Beschreibung von Webformularen standardisiert, um eine Abstraktion für unterschiedliche Plattformen zu schaffen. XForms fügt der Beschreibung der Präsentation (Struktur, Interaktionselemente) eine Spezifikation der verarbeiteten Daten hinzu (Datenmodell), über welche das Verhalten während der Eingabe bestimmt und am Ende das Resultat an einen Server übermittelt werden kann. Die Beschreibung enthält sowohl Elemente auf AUI- als auch der CUI-Ebene. Zwar sind strukturelle Eigenschaften plattformunabhängig beschrieben (z.B. über *groups*), die Präsentation und Datentypen beziehen sich jedoch auf konkrete, grafische Eingabeelemente (z.B. *radio buttons*, *checkboxes* etc.), wodurch sich die Modelle nur eingeschränkt auf andere Modalitäten wie z.B. Sprachinteraktionen übertragen lassen.

UIML [2, 7, 90] ist einer der ersten Ansätze, der ein generisches Modell zur Erzeugung von UIs für beliebige Interaktions- und Technologiekontexte verwendet. UIML erweitert den Fokus auf multimodale UIs und verwendet AUI-Modelle zur Beschreibung von Struktur, Inhalten und Verhalten, ergänzt um Präsentationsregeln. Die Beschreibung wird in die Bereiche Struktur (*structure*), Stil (*style*), Inhalte (*contents*), und Verhalten (*behaviour*) aufgeteilt, deren Inhalte über Vokabularien (*vocabularies*) für spezifische Varianten spezifiziert sind. Die Struktur wird über hierarchisch gruppierte, abstrakte Interaktionselemente beschrieben, für welche Präsentationen (z.B. *position, font, style, color*) und Inhalte (Texte, Bilder etc.) gemäß eines Vokabulars angegeben werden. Eingabe-Ereignisse und resultierende Aktionen werden separat spezifiziert. Durch die Trennung der Aspekte ist die Verwendung unterschiedlicher Vokabularien möglich, wodurch Varianten der Anwendung generiert werden können (z.B. für HTML, VoiceXML und Java Swing).

MANTRA [18] geht bei der Modellierung über die strukturelle Beschreibung hinaus und fügt dem verwendeten AUI-Modell eine Dialogmodellierung hinzu[28]. Die Modellierung erfolgt hier über Interaktionselemente (*UI elements*), deren Gruppierung (*UI composites*) und hierarchische Aggregation. *Composites* können mit Aktions- bzw. Navigationsauslösern (*triggers*) versehen werden, die eine Operation oder die Navigation zu weiteren *Composites* anstoßen. Die Navigation wird über ein Dialogmodell gesteuert, welches die konkreten Ziele bestimmt, wodurch abhängig vom Interaktionskontext der Anwendung unterschiedliche Varianten präsentiert werden können. Zur Erzeugung der finalen UI verwendet MANTRA das im CRF vorgeschlagene Verfeinerungsverfahren und entsprechende Transformatoren.

Bewertung: Die Stärken von UIDLs liegen in der strukturell und funktional vollständigen Modellierung von UIs (A1.1). Dadurch sind Benutzungsschnittstellen grundsätzlich lauffähig generierbar (A3.3). Einschränkungen bestehen jedoch bei der Unterstützung beliebiger Interaktions- und Technologiekontexte (A1.2, A3.2). Insbesondere Ansätze, deren Artefakte auf der CUI-Ebene angesiedelt sind, können nicht für beliebige Kontexte verwendet werden. Auch die auf der AUI-Ebene angesiedelten Ansätze konzentrieren sich auf die Generierung von technischen Varianten für konkrete Anwendungskategorien (z.B. grafische Benutzungsschnittstellen), sodass deren Modelle nicht auf beliebige Interaktions- und Technologiekontexte übertragbar sind.

Keiner der betrachteten Ansätze stellt Lösungen zur einfachen, redundanzfreien Erstellung inhaltlicher Varianten bereit (A1.3). Hier ist lediglich ein *Kopieren und Modifizieren* möglich,

[28] MANTRA ist damit keine reine UIDL, wird jedoch an dieser Stelle aufgeführt, da sich das Modell auf der AUI-Ebene befindet. Der Ansatz könnte auch auf eine taskmodellbasierte Modellierung umgestellt werden [18].

was zu Redundanzen und erhöhten Aufwänden bei Änderungen führt. MANTRA ermöglicht zwar die Beschreibung alternativer Abläufe, die Varianten der dargestellten Sichten müssen jedoch manuell erstellt werden. Zudem existieren nach meinem besten Wissen keine Konzepte zur Integration bestehender Komponenten im Sinne einer Komposition (A.2) und damit zur Generierung kombinierter Anwendungen (A3.1).

Die Artefakte werden in proprietären Notationen beschrieben. Zwar wird meist eine standardisierte Syntax verwendet (z.B. XML), die Inhalte (Semantik) sind jedoch nicht standardisiert (A4.1). Die Zahl der verwendeten Artefakte ist gering. Allerdings fanden sich keine Ansätze, die den Aspekt der gemeinschaftlichen Nutzung behandeln (A4.3).

Task-/Konversationsmodellbasierte Ansätze

Taskmodellbasierte Ansätze gründen auf der Idee, dass UIs über den Dialogfluss und die zu erfassenden Daten beschreibbar sind. Der Dialogfluss wird über Aufgaben und Unteraufgaben (*tasks, subtasks*) modelliert. Die Basis bildet ein **Datenmodell** und ein **Taskmodell**, aus welchen der Dialogfluss abgeleitet wird. Aus dem Dialogfluss und den Informationen des Datenmodells werden AUI-Modelle erzeugt bzw. auf manuell erstellte AUI-Modelle referenziert. Das resultierende AUI-Modell wird dann sukzessive in eine finale UI transformiert. Konversationsmodellbasierte Ansätze sind eine Variante des Ansatzes. Hier werden statt der technischen Taskmodelle Interaktionen als Dialoge mit dem Nutzer (Konversationen) modelliert, die einer natürlichen menschlichen Kommunikation entsprechen [59].

Repräsentative Ansätze sind **MECANO** [181], **UsiXML** [130], **TERESA** [148] und **MARIA** [163]. Sie basieren auf dem CRF und ergänzen die Basismodelle um weitere Modelltypen zur Beschreibung der Präsentation und Navigation, um daraus (ggf. teilautomatisiert) finale Benutzungsschnittstellen für unterschiedliche Modalitäten zu generieren. Konversationsmodellbasierte Ansätze sind in den Arbeiten von **Falb et al.** [59], **Popp et al.** [175] und **Raneburger** [186] dargestellt. **Pleuss et al.** [171, 173] wendet die Konzepte zur teilautomatisierten Generierung von Produktfamilien an. **Gaulke et al.** [76] beschreiben einen Ansatz zur Repräsentation der Modelle in einer nicht-proprietären Form.

MECANO [181] wurde ursprünglich verwendet, um aus einem Domänenmodell eine UI für multiple Plattformen herzuleiten und iterativ manuell zu verfeinern [182]. In der Folge wurde es um Task-, Präsentation-, Dialog-, Design-, und Nutzermodelle erweitert, die jedoch optionalen Charakter haben. Die Interaktion wird bei MECANO durch *Kommandos* gesteuert, die im *Dialogmodell* definiert sind. Das Präsentationsmodell beschreibt die UI auf AUI- oder CUI-Ebene. Das *Designmodell* stellt die Verbindung zwischen den Teilmodellen her. Hier wird z.B. der Zusammenhang zwischen Tasks und Domänenobjekten hergestellt. Zudem wird hier definiert, mit welchen Präsentationskomponenten die Kommandos und Tasks visualisiert werden und die Transitionen zwischen verschiedenen Dialogen einer Anwendung festgelegt. Dieses Vorgehen bietet zwar viele Freiheitsgrade, resultiert jedoch in einer hohen Zahl voneinander abhängender Artefakte.

UsiXML [130] setzt vollständig die Konzepte des CRF um, um multiple Varianten von UIs zu modellieren (s. Abschnitt 2.6.3). Die UI wird über Daten-, Task- und AUI-Präsentationsmodelle beschrieben [1]. Diese Modelle werden um Transformationsmodelle ergänzt [5, 223], welche die automatisierten Verfeinerungen hin zum FUI-Modell beschreiben. Die Transformationen beziehen die hierzu notwendigen Informationen aus einem *Context-of-Use*-Modell.

TERESA [148] verwendet als Basis ein Taskmodell, welches Aktivitäten als *Concurrent Task Tree* (CTT) beschreibt [148]. Durch Verfeinerung der Tasks, wird daraus eine abstrakte UI hergeleitet. Für jeden modellierten Task werden sog. Interaktoren (*interactors*) angegeben, für welche wiederum Präsentationen (*presentations*) für die AUI-Repräsentation spezifiziert werden. Interaktoren werden über Kompositionsoperatoren (*composition operators*) zu Dialogen zusammengesetzt und deren Komponenten und Interaktoren in AUI-Dialogseiten aggregiert. Das resultierende AUI-Modell ist einheitlich in einer eigenen Sprache (*TERESA Abstract User Interface Language*) beschrieben, aus welcher gemäß des CRF-Ansatzes CUI- und FUI-Modelle für multiple Plattformen abgeleitet werden.

MARIA [163] ist eine Weiterentwicklung von TERESA und fokussiert auf die Erstellung für UIs serviceorientierter Anwendungen. Die XML-basierte Beschreibung beinhaltet die Spezifikation von Task-, AUI- und CUI-Modellen. MARIA XML führt ein Datenmodell ein, welches XSD als Typ-Spezifikationssprache verwendet. Diesem wird ein Eventmodell hinzugefügt, welches die Spezifikation von Nutzeraktionen und Reaktionen auf Datenänderungen ermöglicht. Zudem wurde eine spezifisches AUI- und CUI-Metamodell eingeführt, welches eine feingranulare Kontrolle über die Generierungsergebnisse zum Ziel hat. Zur Anbindung von Webservices wurden später Mechanismen eingeführt, welche die Bindung von Service-Aufrufen an Interaktionselemente ermöglichen (z.B. zur Datenbeschaffung, Vorbefüllung und Reaktion auf Nutzerereignisse). In [163] ist außerdem ein erster Ansatz zur Aggregation von MARIA UIs beschrieben.

Falb et al. [59] variieren den taskmodellbasierten Ansatz: hier werden Konversationsmodelle zur Beschreibung verwendet, die Nutzerinteraktionen (*communicative acts*), deren Beziehungen und Abhängigkeiten modellieren. Die resultierenden Modelle beschreiben Anwendungen aus Nutzersicht und weniger technisch als Taskmodelle. Auch **Popp et al.** [175] und **Raneburger** [186] verwenden Konversationsmodelle zur Beschreibung und schlagen in teilweise gemeinsam erstellten Arbeiten Ansätze vor, wie aus diesen Modellen komplett- oder teilautomatisiert finale Benutzungsschnittstellen hergeleitet werden können [60, 176, 186–189].

Pleuss et al. [171, 173] beschreiben erste Lösungen zur Modellierung von Produktfamilien, die Konzepte zur teilautomatisierten Generierung von Varianten beinhalten. Hierbei dienen Taskmodelle als Grundlage, die in einem manuellen Schritt gruppiert und auf AUI-Modellebene umstrukturiert werden können [173]. **Tran et. al** [223] automatisieren die Erstellung von Varianten durch die Auswahl von Tasks für eine Variante aufgrund von Gewichtungen, die vom Modellierer für einzelne Bereiche des Taskmodells angegeben werden können.

Die Modellierung erfolgt in den dargestellten Ansätzen über proprietäre Modelle. **Gaulke et al.** [76] präsentieren einen Ansatz, der Task- und Datenmodelle mit semantischen Technologien beschreibt. Dabei werden diese Modelle in einer universellen Form über OWL-Ontologien und damit in einer universellen, nicht-proprietären Form repräsentiert.

Bewertung: Die Stärken liegen in der flexiblen Modellierung der Abläufe, die zu einer umfassenden Kontrolle der Frageflüsse führen. Einschränkungen weisen die Ansätze jedoch in der geforderten Vollständigkeit insbesondere in der Modellierung des Verhaltens auf. So sind komplexere Dialogflüsse zwar modellierbar, Operationen mit Einfluss auf das Verhalten (z.B. bedingte Anzeige von Dialogbereichen) sind nicht im Fokus der Modellierung bzw. werden auf die AUI-Ebene verlagert (A1.1).

Die Erzeugung technischer Varianten wird insbesondere von den aktuelleren Ansätzen unterstützt (A1.2). Eine vollständige Generierung ist möglich (A3.2, A3.3). Zur redundanzfreien Beschreibung und Generierung inhaltlicher Varianten sehen die Ansätze hingegen keine vollständig automatisierbaren Lösungen vor (A1.3, A3.1). Die Ansätze erfordern meist die manuelle Erstellung inhaltlicher Varianten durch Angabe von spezifischen Task- und/oder Präsentationsmodellen (*Kopieren und Modifizieren*). Pleuss et al. [173] schlagen zwar einen Ansatz für Produktfamilien vor, der jedoch weiterhin die manuelle Modellierung redundanter Taskmodellvarianten erfordert.

Zur Komposition von Modellen im Sinne der Anforderungen unter (A.2) existieren nach meinem besten Wissen keine Lösungen. Zu Einbindung bestehender Modelle an beliebigen Stellen eines Frageflusses (A2.1) existieren in MARIA zwar Ansätze zur Aggregation von Anwendungen, die eingebundenen Komponenten können jedoch nicht in Struktur und Verhalten angepasst werden (A2.2).

Die Zahl der zu modellierenden Artefakte ist höher als bei UIDLs, was deren Verteilung und Wiederverwendung erschwert. Es werden mehrere, voneinander abhängige Modelle benötigt, die unterschiedlichen Abstraktionsstufen zuzuordnen sind (Daten-, Task-, AUI-/CUI-Präsentationsmodelle). Die Artefakte sind zudem in proprietärer Form beschrieben, was eine gemeinschaftliche Nutzung einschränkt (A4.1, A4.3). Mit den Arbeiten von Gaulke et al. [76] zur ontologischen Beschreibung besteht eine erste Lösung dieses Problems. Konzepte zur direkten Anbindung von Diensten liegen ebenfalls nicht im Fokus der Ansätze (A4.2). Hier bestehen zwar erste Ansätze in MARIA, die jedoch auf SOAP-basierten WebServices beruhen und damit nicht universell anwendbar sind.

Datenmodellbasierte Ansätze

Die hierunter zusammengefassten Ansätze folgen der Idee, dass UIs direkt aus den Daten einer Anwendung herleitbar sind. Sie erzeugen UIs direkt aus einem domänenspezifischen Datenmodell, für welches Sichten (*views*) automatisch abgeleitet bzw. manuell erstellt werden.

In **JANUS** [9, 10] wird die Benutzungsschnittstelle einzig aus den Klassen (Entitäten) eines objektorientierten Domänenmodells abgeleitet. Es werden keine darüber hinausgehenden Modelle verwendet. Für die nicht abstrakten Entitäten des Modells werden Sichten generiert, die über eine Navigation verbunden sind. Diese ergibt sich aus den Beziehungen zwischen den Klassen. Für jedes Attribut der Entitäten werden Interaktionselemente anhand der Typinformationen erzeugt. Das Ergebnis ist ein CUI-Modell in Form einer textuellen Beschreibung. Darüber hinaus werden in JANUS Datenbankschemata zur Persistierung der Daten und Operationen erzeugt, die das Lesen, Schreiben und Modifizieren der Entitäten in einer Datenbank ermöglichen. Der Ansatz beschränkt sich auf die generische Erzeugung von CRUD-Anwendungen (*create, read, update, delete*), die zur Manipulation der modellierten Datenstrukturen genutzt werden können.

Die frühen Stadien von **MECANO** [182] ähneln denen von JANUS. Hier werden ebenfalls Darstellungseinheiten direkt aus Domänenklassen erzeugt. Die Eingabeelemente werden aufgrund der Attribute der Domänenklassen erzeugt und die Navigationsstruktur resultiert aus den Beziehungen zwischen den Klassen.

Auch in **GENIUS** [112] ist das Domänenmodell die Ausgangsbasis. Die *Sichten* werden hier jedoch manuell erstellt und die Navigation über sog. *Dialognetze* spezifiziert. Im Gegensatz zu JANUS können damit aufgabenspezifische Dialogstrukturen erstellt werden. Die Herleitung der FUI erfolgt automatisch durch ein regelbasiertes System, welches Design-Regeln für einen konkreten Technologiekontext enthält und zu einer konsistenten, einheitlichen Dialogstruktur führt.

In den vergangenen Jahren entstanden zudem Ansätze, die im Umfeld von Web Services entwickelt wurden. Sie basieren auf Datenmodellen, die in Dienstschnittstellen spezifiziert werden und leiten daraus meist sehr einfach gehaltene UIs ab. **Lay et al.** [128] präsentieren beispielsweise einen Ansatz zur Erzeugung von UIs basierend auf XML-Schemabeschreibungen. In **AbuJarour et al.** [3] dienen die in WebService-Beschreibungen (WSDL) enthaltene Informationen zur Herleitung einer UI. In **Khushraj et al.** [115] dienen Dienstbeschreibungen als Grundlage, die mit semantischen Technologien über RDF-/OWL-Ontologien spezifiziert sind.

Bewertung: Die Stärke liegt im hohen Abstraktionsgrad der Modelle, in der geringen Anzahl zu erstellender Artefakte und deren Einfachheit. Damit geht jedoch ein geringer Funktionalitätsumfang einher (A1.1). So können zwar Daten und Sichten angegeben, jedoch das dynamische Verhalten nicht oder nur eingeschränkt modelliert werden (z.B. können in JANUS lediglich automatisch erzeugte CRUD-Operationen verwendet werden).

Aufgrund des höheren Abstraktionsgrades der Modelle sind sie zur Erstellung technischer Varianten geeignet (A1.2, A3.2), sofern ausreichend Informationen zu den Eigenschaften der Daten im zugrunde gelegten Modell vorliegen. Nach meinem besten Wissen existieren jedoch keine Ansätze zur Modellierung und Generierung inhaltlicher Varianten (A1.2, A3.1). Ebenso wenig werden die Kombinierbarkeit und Komposition berücksichtigt (A.2).

Die Zahl der notwendigen Artefakte ist gering. Sie sind vorwiegend in einer proprietären Form beschrieben. In den XML-basierten Datenbeschreibungen wird jedoch eine standardisierte Syntax verwendet, deren Semantik (z.B. Typen, Restriktionen etc.) über XML-Schema festgelegt ist. Die Arbeiten von **Khushraj et al.** [115] zeigen die Nutzung von ontologiebasierten Beschreibungen, die universell einsetzbar sind. Insofern sind die Artefakte eingeschränkt verteilbar (A4.1). Die WebService-basierten Ansätze sind erste Lösungen der Serviceanbindung, beschränken sich jedoch auf konkrete Technologien. Daher sind sie nur begrenzt übertragbar (A4.2). Eine Untersuchung zur Wiederverwendung in einem dezentralen Ökosystem wurde in der Literatur nicht gefunden (A4.3).

Funktionale UI Spezifikationen

In diese Kategorie fallen insbesondere die Arbeiten von Nichols et al. (**PUC**, *Personal Universal Controller*) [150, 151] und **SUPPLE** [73, 74]. Die Modelle beschreiben Struktur und Typen der von der Anwendung verarbeiteten Informationen, deren Gruppierung, Zusammenhänge und ausführbare Operationen (*commands*) auf dem Modell. Dies wird als *Funktionalität* aufgefasst, die dem Nutzer präsentiert werden kann. Was zu einem Zeitpunkt konkret angezeigt - und damit als Funktionalität angeboten - wird, hängt vom Zustand der Daten ab. Die funktionale UI Spezifikation stellt eine spezielle Form eines datenmodellbasierten Ansatzes dar. Hierbei wird jedoch kein Domänenmodell, sondern ein **Modell der von der Anwendung zu präsentierenden Daten** als Grundlage verwendet. Eine Instanz des Modells zur Laufzeit repräsentiert den Zustand der Anwendung und bestimmt die dargestellten

Inhalte. Operationen verändern diesen Zustand und bestimmen damit das Verhalten der Anwendung.

Die Generierung der Benutzungsschnittstelle (Struktur, Interaktionselemente) erfolgt aufgrund der im Datenmodell enthaltenen Informationen (Typ, Einschränkungen etc.). Ergänzend können Materialien angegeben werden, die für die finale Darstellung verwendet werden sollen (z.B. Texte für Labels und Layoutinformationen). Zur Erzeugung der FUI werden plattformspezifische Vorlagen für Interaktionselemente (*templates*) verwendet [150], welche zu einer Benutzungsschnittstelle zusammengefügt werden.

Der funktionale Ansatz unterscheidet sich signifikant von den bisher beschriebenen Ansätzen. Es wird bewusst auf die explizite Modellierung von Tasks oder Abläufen verzichtet [151] und stattdessen im Modell Reaktionen auf Zustandsänderungen der Instanzdaten beschrieben. Zudem erfolgt keine explizite Modellierung der Benutzungsschnittstelle auf AUI- oder CUI-Ebene. Diese wird aus der weitaus abstrakteren Beschreibung der Daten abgeleitet. Dies ermöglicht die automatische Erzeugung für beliebige Interaktions- und Technologiekontexte.

Bewertung: Die Stärken liegen in der Einfachheit der Modelle und der bezüglich der Generierung vollständigen Beschreibung. Aufgrund des hohen Abstraktionsgrades der Modelle eigenen sie sich zur Erzeugung technischer Varianten (A1.1, A1.2, A3.2, A3.3).

Inhaltliche Varianten können prinzipiell ebenfalls erstellt werden. SUPPLE bietet ein entsprechendes Konzept, welches allerdings komplex in der Umsetzung ist. Varianten werden hier zur Laufzeit aus dem Nutzungskontext und Heuristiken ableitet [73]. Hierzu werden variantenspezifische Herleitungsalgorithmen (*cost functions*) verwendet, welche die UI erzeugen. Dies beschränkt die Nutzbarkeit, da keine deklarative Variantenbeschreibung möglich ist. Für neue Varianten müssen ggf. die Herleitungsalgorithmen angepasst werden (A1.3, A3.1).

Da die Beschreibung von UIs kompakt und vollständig in technologieneutralen, datenzentrierten Modellen erfolgt, könnten diese prinzipiell zu vollständig generierbaren Anwendungen im Sinne einer *Aggregation* zusammengesetzt werden (A2.1, A2.3). Jedoch wurde dies in der Literatur zu den Ansätzen nicht näher untersucht. Daher existiert in den Ansätzen auch kein Konzept zur inhaltlichen Modifikation von Komponenten (A2.2), das zur Anpassung an neue Nutzungen verwendet werden könnte. Eine Komposition im geforderten Umfang ist daher mit den bestehenden Lösungen nicht möglich.

Ebensowenig wurde der Aspekt der Wiederverwendung und gemeinschaftlichen Nutzung für diese Ansätze in der Literatur untersucht (A4.3). Die Modelle sind proprietär (A4.1) und es existieren keine Ansätze zur Anbindung von Diensten (A4.2).

Web-Engineering Ansätze

Parallel zu den genannten Ansätzen entstanden Lösungen zur modellgetriebenen Entwicklung von Web-Anwendungen. Deren Ziel liegt in der umfassenden Modellierung kompletter Applikationen von der Datenhaltung bis hin zur Benutzungsschnittstelle. Hierbei werden komplette Websites als Anwendungen aufgefasst und über ihre Daten, Abläufe und webbasierte Oberflächen beschrieben. Die verwendeten (oft manuell erstellten) Modelle erstrecken sich auf der Daten-, Taskmodell-, CUI-Ebene bis hin zu FUI-Elementen, aus welchen ein ausführbares FUI hergeleitet wird.

Repräsentative Ansätze sind **UWE** [120, 121, 204], **OO-H** [79], **WebML** [28] und **MDWEnet**, [226] die fachliche Daten-, UseCase- und Zustandsmodelle als Ausgangsbasis verwenden.

UWE (*UML-based Web Engineering*) [120, 121, 204] ist ein iterativer Ansatz zur umfassenden Modellierung von Web-Anwendungen entlang der Phasen des Entwicklungsprozesses. Der Ansatz wendet die MDA-Prinzipien der OMG an (vgl. Abschnitt 2.6.2) und schlägt eine Menge von Domänen-, PIM-, and PSM-Modellen vor, die als Ergebnis aus der Analyse-, Design- und Implementierungsphase resultieren. Die funktionalen Anforderungen werden über Daten- und UseCase- und Zustandsmodelle erfasst [119]. UWE verwendet diese als Grundlage zur manuellen Erstellung von AUI-, CUI- und FUI-Modellen für Inhalt, Navigation, Abläufe, Darstellung sowie Adaptivitätsverhalten. Transformatoren generieren daraus PIM-Modelle, die in weiteren Schritten manuell verfeinert werden können. Die Designmodelle werden abschließend zusammengeführt und in eine plattformspezifische UI transformiert.

In **OO-H** (*Object Oriented Hypermedia*)[79] werden Anwendungen über statische und dynamische Aspekte modelliert. Neben dem Datenmodell existiert ein Navigations- und ein abstraktes Präsentationsmodell zur Beschreibung der UI. Das Navigationsmodell wird auf Grundlage eines UseCase- und Domänenmodells erstellt. Ein erstes, abstraktes Präsentationsmodell kann hieraus automatisiert erstellt werden. Es beschreibt die Struktur, Nutzerinteraktionen, Navigation, Seitenaufteilung und die Präsentation. Dieses wird manuell verfeinert und daraus ein FUI automatisiert für eine spezifische Plattform hergeleitet.

Einen ähnlichen Ansatz verfolgen Ceri et al. mit **WebML** (*Web modelling Language*) [28], die einen Entwicklungsprozess und resultierende Artefakte für *datenintensive Anwendungen* beschreiben. Hierbei werden Web-Anwendungen über ein *Daten-, Hypertext- und Content Management-Modell* spezifiziert. In einem zusätzlichen Modell (*Advanced Hypertext-Modell*) wird das Verhalten der Anwendung modelliert.

Die Ansätze ähneln sich in vielen Bereichen, weshalb sich die Autoren zusammengeschlossen haben und eine Initiative zur Konsolidierung insbesondere von UWE, OO-H und WebML ins Leben riefen (**MDWEnet**, [226]). Das Ziel liegt in der Angleichung der Ansätze, deren Interoperabilität und Verbesserung der Werkzeuge zur modellgetriebenen Entwicklung webbasierter Anwendungen.

Bewertung: Obwohl diese Ansätze sehr detailliert die Benutzungsschnittstellen webbasierter Anwendungen beschreiben können, ist hier eine hohe Zahl voneinander abhängiger Artefakte auf unterschiedlichen Abstraktionsstufen (Daten und Abläufe, AUI-, CUI-Modelle, FUI-Elemente) manuell zu erstellen. Lediglich die abschließende Erzeugung der zusammengesetzten FUI-Artefakte ist vollständig automatisiert möglich und führt zu einer ausführbaren UI (A3.3). Die Modelle sind vollständig, erfordern jedoch einen hohen Aufwand in Erstellung und Wartung aufgrund der Komplexität und Abhängigkeiten (A1.1). Sowohl technische als auch inhaltliche Varianten müssen manuell modelliert werden (A1.2, A1.3) und sind nicht aus einem Basismodell ableitbar. Kompositionen im Sinne (A.2) sind in diesen Ansätzen nicht vorgesehen. Aufgrund der hohen Artefaktzahl, deren Abhängigkeiten untereinander und zu Webtechnologien ist deren Wiederverwendbarkeit im Sinne (A.4) nicht gegeben.

3.2.3 Ungelöste Problemstellungen

Tabelle 3.2 fasst die Bewertungen der betrachteten Kategorien zusammen. In den Zeilen sind Anforderungen an eine Lösung in den Bereichen (A.1)-(A.4) aufgeführt (vgl. Abschnitt 3.1). In den Spalten ist aufgetragen, inwieweit die Ansätze der betrachteten Kategorien diese Anforderungen erfüllen. Am Ende der Tabelle sind als *Generelle Bewertungskriterien* die Komplexität und Technologieabhängigkeit der Modellierung aufgeführt. Sie sind ein Indikator für die grundsätzliche Praktikabilität des Ansatzes. Eine geringe Komplexität weist auf eine einfachere Erstellung und Wartung der Modelle hin. Eine geringe Technologieabhängigkeit impliziert die Anwendbarkeit des Ansatzes auf multiple Interaktions- und Technologiekontexte.

	Anforderung	UIDL	Task-/ Konversations-modell	Daten-modell	Funktionale Spezifikation	Web Engineering
A.1	**Modellierung von Benutzungsschnittstellen**					
(A1.1)	Vollständigkeit des Modells	●	◑	◑	●	●
(A1.2)	Eignung zur Erzeugung technischer Varianten	◑	●	◕	●	◌
(A1.3)	Eignung zur redundanzfreien Beschreibung inhaltlicher Varianten	◌	◌	◌	◑	◌
A.2	**Komposition von Benutzungsschnittstellen**					
(A2.1)	Freie Kombinierbarkeit von Modellen	◌	◔	◌	◔	◌
(A2.2)	Anpassbarkeit der Komponenten an ihr Umfeld	◌	◌	◌	◌	◌
(A2.3)	Vollständigkeit der Komposition	◌	◔	◌	◔	◌
A.3	**Verfahren zur Generierung von Varianten**					
(A3.1)	Herleitung inhaltlicher Varianten und Kompositionen	◌	◌	◌	◑	◌
(A3.2)	Generierung technischer Varianten	◑	●	●	●	◌
(A3.3)	Ausführbarkeit in Laufzeitumgebung	●	●	●	●	●
A.4	**Wiederverwendung in einem Service-Ökosystem**					
(A4.1)	Verwendung einer universell interpretierbaren Repräsentation	◑	◔	◑	◌	◌
(A4.2)	Bereitstellung erfasster Daten für Dienste im Service-Ökosystem	◌	◌	◑	◌	◌
(A4.3)	Verteil- und Verwendbarkeit in einer dezentralen Infrastruktur	◌	◌	◌	◌	◌
	Generelle Kriterien zur Bewertung der Praktikabilität					
	Komplexität der Modellierung: Zahl zu modellierender Artefakte	gering	mittel/hoch	gering	gering/mittel	hoch
	Technologieabhängigkeit der Modellierung	hoch	mittel/hoch	gering	gering	hoch

erfüllt	●
teilweise erfüllt	◔◑◕
nicht erfüllt	◌

Tabelle 3.2. Erfüllung der Anforderungen durch bestehende Ansätze

Die Tabelle verdeutlicht, dass keiner der betrachteten Ansätze die in Abschnitt 3.1 gestellten Anforderungen vollständig erfüllt. Die Bewertung der Kategorien anhand der *Generellen Bewertungskriterien* (Tabelle 3.2, unten) zeigt, dass die Modelle insbesondere bei UIDL, Task- / Konversationsbasierten und Web-Engineering-Ansätzen sowohl technologiespezifisch als auch komplex in der Anwendung sind. UIs werden hier über mittlere bis hohe Zahl voneinander abhängiger Artefakte modelliert, die bei Erstellung und Wartung konsistent gehalten werden müssen. Die Ansätze beschränken sich zudem auf spezifische Interaktions- und Technologiekontexte. Dies erschwert die redundanzfreie Modellierung inhaltlicher Varianten und deren Generierung für beliebige technische Umgebungen.

Die Tabelle zeigt zudem, dass sich die bestehenden Ansätze auf die Modellierung und automatische Erzeugung *technischer Varianten* konzentrieren. So sind die Anforderungen (A1.1) und (A1.2) weitgehend erfüllt und meist eine vollständig automatisierte Herleitung lauffähiger UIs möglich (A3.2, A3.3). Hinsichtlich der Eignung zu **redundanzfreien Spezifikation und Generierung inhaltlicher Varianten bestehen jedoch Lücken** bei der Mehrzahl der Ansätze (A1.3, A3.1). Die Problemstellungen der **Komposition (A.2) und Wiederverwendbarkeit in einem Service-Ökosystem (A.4) sind weitgehend ungelöst**. Kein Ansatz besitzt eine den Anforderungen genügende Möglichkeit zur Komposition (A2.1, A2.2, A2.3, A3.1). Die Mehrheit der Ansätze ist zudem gemäß der Anforderungen unter (A.4) nicht zur gemeinschaftlichen Nutzung geeignet. Sie basieren vorwiegend auf proprietären Formaten und sehen kein Konzept für die Dienstanbindung vor. Sofern hierfür erste Lösungen existieren, erfüllen sie die Anforderungen unter (A.1)-(A.3) nur unzureichend. Sie eignen sich daher nicht zur Erstellung von Shared UIs (A.4).

Nach Tabelle 3.2 erfüllen die datenmodellbasierten Ansätze und funktionalen UI Spezifikationen am ehesten die gestellten Anforderungen. Aufgrund der geringen Komplexität der Modellierung sowie der geringen Technologieabhängigkeit besteht hier das größte Potential zur Ergänzung der fehlenden Konzepte.

3.3 Lösungsansatz, Architektur und Realisierungskonzepte

Zur Lösung der offenen Problemstellungen wird ein modellgetriebener Ansatz verfolgt, der mit einer geringen Anzahl technologieneutral gehaltener Artefakte die Anforderungen unter (A.1)-(A.4) durchgängig erfüllt.

In Abschnitt 3.3.1 wird die Lösungsarchitektur und darin enthaltene Elemente zur Umsetzung der Modellierung und der automatischen Erzeugung dargestellt. Er fokussiert auf die offenen Problemstellungen bestehender Ansätze hinsichtlich der Modellierung inhaltlicher Varianten und Kompositionen (A.1, A.2) und ergänzt bestehende Generierungsansätze um diese Aspekte (A.3). In Abschnitt 3.3.2 wird ein Ansatz und eine Lösungsarchitektur aufgezeigt, welche die Modelle im Rahmen eines verteilten Dienst-Ökosystems nutzbar machen. Er fokussiert damit auf die unter (A.4) gestellten Anforderungen und offenen Problemstellungen. In Abschnitt 3.3.3 werden ergänzend die Realisierungskonzepte dargestellt, mit denen der Lösungsansatz in der Arbeit umgesetzt wird und zeigt auf, wie hierdurch die offenen Problemstellungen gelöst werden.

3.3.1 Modellierung und Generierung von Benutzungsschnittstellen

Der verfolgte Lösungsansatz greift die **Grundidee der datenmodellbasierten und funktionalen** Ansätze auf und passt diese den erweiterten Anforderungen an. Er besteht aus vier Kernelementen, die in Abbildung 3.2 dargestellt sind. Hierbei wird zwischen Modell- und funktionalen Elementen unterschieden. Die Modellelemente beschränken sich auf zwei manuell zu erstellende Artefakte:

- ein zentrales **Anwendungsmodell** ①
- mehrere (optionale) **Variantenbeschreibungen** ②

Die funktionalen Elemente des Ansatzes sind hingegen vollständig automatisierte Verfahren. Sie bestehen aus einem

- **Verfahren zu Generierung** von Benutzungsschnittstellen ④
- **Verfahren zur Aufbereitung eingegebener Daten** ⑤

Das zentrale Element bildet das **Anwendungsmodell** ①. Der Lösungsansatz besteht darin, hierfür ein *Gesamtmodell der zur erfassenden Daten* einer Anwendung als zentrales Artefakt zugrunde zulegen. Dieses wird um *Informationen angereichert*, die zur vollständigen Generierung einer Benutzungsschnittstelle notwendig sind. Um auf multiple Kontexte anwendbar zu sein, abstrahiert das Modell dabei von konkreten Technologiespezifika.

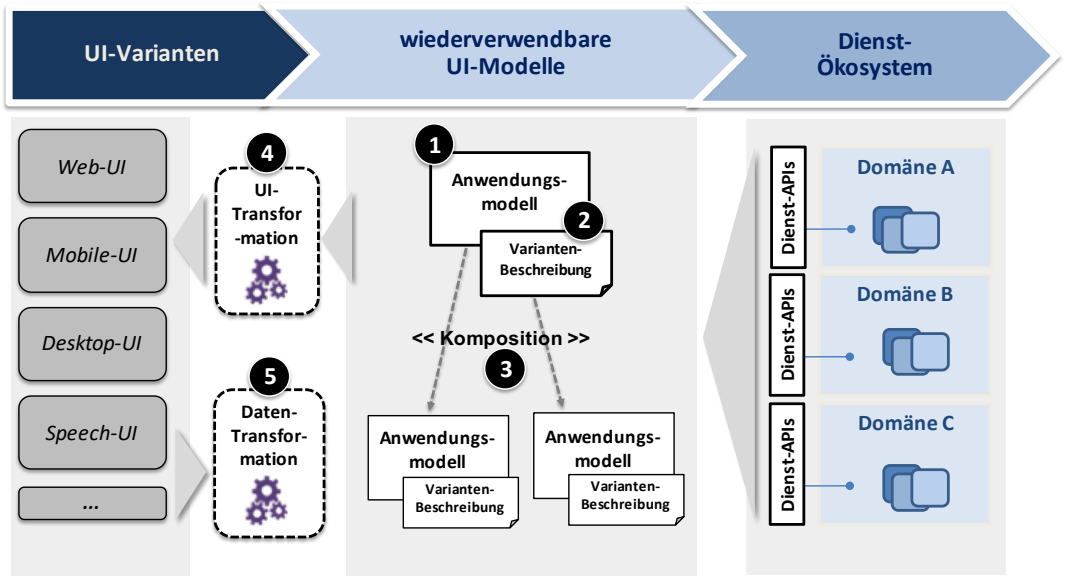


Abbildung 3.2. Elemente der Lösungsarchitektur

In einem **Dienst-Ökosystem** bilden die Daten der anzubindenden Dienste die Grundlage für das Anwendungsmodell. Diese sind durch die Dienst-Schnittstellen (Abbildung 3.2, *Dienst APIs*) spezifiziert. Das Anwendungsmodell ergänzt diese um Informationen zu Struktur, Typisierung und Verhalten und schafft damit ein vollständiges, technologieneutrales Modell gemäß den Anforderungen (A1.1) und (A1.2).

Die Bildung inhaltlicher Varianten erfolgt über multiple **Variantenbeschreibungen** ②, die auf dem Anwendungsmodell basieren. Der Lösungsansatz besteht darin, *Unterschiede des Inhalts und Verhaltens* einer Variante zum zugrunde gelegten Anwendungsmodell zu be-

schreiben. Aus dem Anwendungsmodell und der differentiellen Beschreibung kann ein Anwendungsmodell für die Variante abgeleitet werden, welches wiederum vollständig ist und den Anforderungen (A1.3) genügt.

Zur Erstellung von **Kompositionen** können bestehende Modelle als Komponenten in Anwendungsmodellen referenziert und eingebunden werden ③. Es entsteht ein kombiniertes Anwendungsmodell, das wiederum vollständig ist. Um die Komposition einer neue Nutzung anzupassen, können Variantenbeschreibungen angegeben werden, die das resultierende Modell in Umfang und Verhalten modifizieren.

Da ein Anwendungsmodell die Benutzungsschnittstelle vollständig beschreibt, können die weiteren Schritte zur Herleitung der konkreten Ausprägung vollständig automatisiert werden. Hierzu wird ein generisches Verfahren zur Erzeugung von Benutzungsschnittstellenvarianten verwendet (**UI-Transformation** ④). Dieses erzeugt aus dem (ggf. kombinierten) Anwendungsmodell und der Variantenbeschreibung ein Variantenmodell und nutzt die enthaltenen Informationen zur Herleitung einer Benutzungsschnittstellen für konkrete Interaktions- und Technologiekontexte (A.3).

Um die erfassten Daten (**Instanzdaten**) nutzen zu können, müssen diese von der Benutzungsschnittstelle in geeigneter Form bereitgestellt werden. Abhängig von der konkreten Implementierung des Dienstes variiert hierbei das Format, in welchem die Instanzdaten übergeben werden müssen. Da das Anwendungsmodell inhaltlich auf den Daten von Dienst-Schnittstellen basiert, kann eine entsprechende Repräsentation automatisch erzeugt werden. Hierfür ist im Lösungsansatz ein generisches Verfahren zur **Daten-Transformation** ⑤ vorgesehen, welches die erfassten Daten auf die geforderten Eingaben der Dienst-Schnittstelle abbildet.

3.3.2 Gemeinschaftliche Nutzung in einem Dienst-Ökosystem

Mit den o.g. Mitteln zur Modellierung, Generierung und Instanzdatentransformation lässt sich eine Verteilung und gemeinschaftliche Nutzung mit einfachen Mitteln erreichen. Abbildung 3.3 zeigt schematisch die Verwendung der Artefakte in einem verteilten Systemkontext für ein Dienst-Ökosystem.

Der Lösungsansatz besteht darin, die Anwendungsmodelle in einer **nicht-proprietären, universell interpretierbaren Repräsentation** zu beschreiben, die als gemeinsame Sprache von Teilnehmern in einem Ökosystem genutzt wird. Diese Repräsentation wird verwendet, um Anwendungsmodelle dezentral z.B. in frei zugänglichen Repositories oder als Link im World Wide Web zur Verfügung zu stellen ①. Die Bereitstellung kann durch beliebige Anbieter erfolgen, welche Anwendungsmodelle für Dienste eines Dienst-Ökosystems als Komponenten zur gemeinschaftlichen Nutzung anbieten.

Eine Anwendung (**Client-Anwendung**, ②) kann diese Komponenten aus den Repositories beziehen, ggf. inhaltliche Varianten und Kompositionen bilden und daraus Benutzungsschnittstellen erzeugen. Da die Anwendungsmodelle in einer standardisierten Form vorliegen, kann die Client-Anwendung hierzu auf **UI-Transformatoren** ③ zurückgreifen, die das o.g. Verfahren zur Generierung implementieren. Dies ermöglicht es der Client-Anwendung, UIs für beliebige Interaktions- und Technologiekontexte zu erzeugen und dabei bestehende Verfahren wiederzuverwenden.

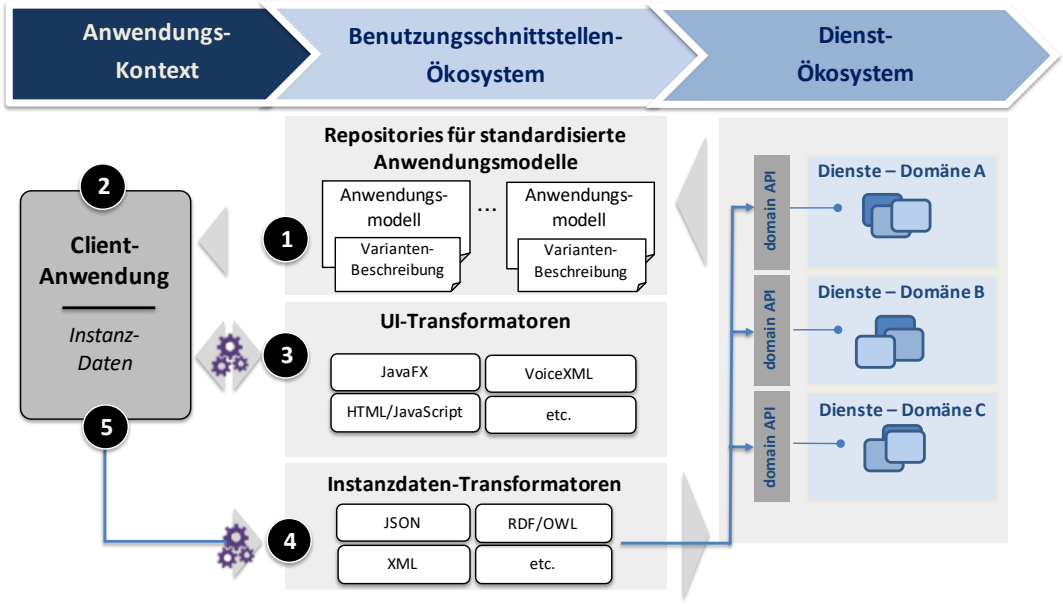


Abbildung 3.3. Anwendung in einem Dienst-Ökosystem

Die von den eingebundenen Komponenten erfassten Daten können in einem weiteren Schritt automatisiert in ein zur weiteren Verarbeitung geeignetes Format überführt werden. Hierzu kann die Client-Anwendung **Instanzdaten-Transformatoren** ④ wiederverwenden, die das o.g. Verfahren zur Daten-Transformation implementieren. Die Client-Anwendung kann die Ergebnisse abschließend an konkrete Dienst-Implementierungen weiterleiten ⑤.

Der vorgeschlagene Lösungsansatz führt zu einem **Benutzungsschnittstellen-Ökosystem**, welches **parallel zum Dienst-Ökosystem** existiert. Anwendungsmodelle für Dienste können darin frei wiederverwendet und automatisch eine an die Umgebung angepasste Benutzungsschnittstelle generiert werden. Die eingegebenen Daten können an konkrete Dienst-Implementierungen im Dienst-Ökosystem übergeben und von diesen verarbeitet werden.

3.3.3 Realisierungskonzepte und deren Implikationen

Dieser Abschnitt beschreibt die konkreten Konzepte zur Umsetzung des im vorangegangenen Abschnitt beschriebenen Lösungsansatzes. Während in den vorangegangenen Abschnitten die Lösungsarchitektur dargestellt wurde, wird hier das Lösungsdesign dargestellt.

Modellierung von Benutzungsschnittstellen

Der Kern des dargestellten Lösungsansatzes besteht in der Verwendung eines **technologieneutralen, datenzentrierten Anwendungsmodells**. In Anlehnung an die Arbeiten von Nichols et al. [150, 151] basiert dieses auf den zu erfassenden Daten der Anwendung und ergänzt Informationen zur Semantik, Struktur und Verhalten, die zur automatisierten Erstellung von Benutzungsschnittstellen notwendig sind. Diese Eigenschaften werden unabhängig von einem Interaktions- und Technologiekontext beschrieben. Das Modell setzt dabei die Anforderungen (A1.1) und (A1.2) um.

Die Beschreibung der Modellinhalte erfolgt in Form eines **Metamodells**, das unabhängig von einer konkreten Notation ist. Es bildet die Grundlage zur Spezifikation verschiedener

Modellierungssprachen. Für Modellierungssprachen wird eine deklarative Repräsentation vorgeschlagen, die alle Informationen des Anwendungsmodells **in einem einzigen Artefakt** beschreibt. Dies unterstützt die einfache Verteilbarkeit der Modelle als Komponenten. In der Arbeit wird hierfür exemplarisch eine Notation in Form einer *Domain Specific Language* (*mimesis.model.DSL*) vorgestellt.

Im Vergleich zur Mehrzahl der bestehenden Ansätze verschiebt sich der Fokus von der expliziten Beschreibung einer UI hin zur Datenmodellierung, die auf eine explizite Modellierung für konkrete Technologien, Interaktionsformen oder Abläufe verzichtet. Hierdurch wird erreicht, dass die Modelle auf beliebige Interaktions- und Technologiekontexte anwendbar sind. Durch die Verwendung eines deklarativen und in sich vollständigen Modellartefakts wird einerseits die Komplexität der Modellierung reduziert, andererseits werden wiederverwendbare und verteilbare Komponenten geschaffen. Durch die Beschreibung der Modellinhalte in einem Metamodell wird dem Problem einer proprietären Modellierung begegnet. Der Ansatz ermöglicht die Nutzung verschiedener, inhaltlich äquivalenter Notationen, die standardisiert und gemeinschaftlich nutzbar sind.

Modellierung inhaltlicher Varianten

Der Lösungsansatz zur Modellierung inhaltlicher Varianten basiert darauf, dass diese Teilmengen der Daten des Anwendungsmodells verarbeiten und ein dem Nutzungskontext angepasstes Verhalten aufweisen. Das Realisierungskonzept besteht darin, diese Unterschiede zum Anwendungsmodell in einer Variantenbeschreibung zu modellieren. Diese beschreibt einerseits die Auswahl relevanter Elemente des Anwendungsmodells, andererseits notwendige Modifikationen an den ausgewählten Elementen.

Die Spezifikation der Modellinhalte erfolgt analog dem Metamodell für Anwendungen in Form eines **Metamodells**, aus welchem konkrete Notationen abgeleitet werden können. Für Modellierungssprachen zur Umsetzung des Metamodells wird eine deklarative Repräsentation vorgeschlagen. In der Arbeit wird hierfür exemplarisch eine Notation in Form einer *Domain Specific Language* (*mimesis.variant.DSL*) vorgestellt.

Durch die differentielle Beschreibung wird eine redundanzfreie Variantenerstellung erreicht, wie sie in (A1.3) gefordert wird. Die erfassten Daten werden im Anwendungsmodell einmalig beschrieben und in der Variantenbeschreibung lediglich zur Beschreibung der Variante referenziert. Der vorgeschlagene Ansatz unterscheidet sich damit von den bestehenden Ansätzen, in denen Varianten durch *Kopieren und Modifizieren* erzeugt werden können.

Komposition von Anwendungsmodellen

Der dargestellte Lösungsansatz für Kompositionen basiert darauf, dass bestehende Anwendungsmodelle zu neuen Anwendungen kombiniert werden können, sofern die Komponenten vollständig spezifiziert sind. Das Realisierungskonzept besteht in der **Erweiterung des Anwendungsmodells um Integrationspunkte**, an denen bestehende Modelle als Komponenten referenziert werden. Während der Generierung werden die Referenzen aufgelöst und die Komponenten als Bestandteil des Anwendungsmodells integriert. Da die Komponenten vollständige Anwendungsmodelle sind, ist das Ergebnis wiederum ein vollständiges Anwendungsmodell gemäß (A1.1). **Notwendige Anpassungen an die neue Nutzung** von Komponenten werden als Variantenbeschreibung des resultierenden Modells vorgenommen. Mittels

dieses Ansatzes werden die Anforderungen unter (A.2) umgesetzt. Bestehende Komponenten können beliebig kombiniert (A2.1) und angepasst werden (A2.2). Es entsteht dadurch ein Anwendungsmodell, aus dem Benutzungsschnittstellen generiert werden können (A2.3).

Der vorgeschlagene Ansatz unterscheidet sich von bestehenden Ansätzen durch Modifizierbarkeit der Komponenten durch die nutzende Anwendung. Anpassungen können deklarativ als Variantenbeschreibung für einen konkreten Anwendungsfall vorgenommen werden, ohne die Komponenten selbst zu modifizieren.

Vollständig automatisierte Generierung von Benutzungsschnittstellen

Der Lösungsansatz zur automatischen Generierung nutzt die Eigenschaft, dass dialogbasierte Benutzungsschnittstellen einen hohen Standardisierungsgrad aufweisen (vgl. Abschnitt 2.2.2) und sich daher beliebige technische Varianten automatisiert aus einem Anwendungsmodell gemäß (A1.1) ableiten lassen.

Das Realisierungskonzept besteht in der Umsetzung eines Verfahrens, welches auf dem CAMELEON-Referenzframework [25] basiert (vgl. Abschnitt 2.6.3). Es wird ein Verfahren angegeben, das ohne manuellen Eingriff aus den o.g. Artefakten eine Benutzungsschnittstelle ableitet. Der CAMELEON-Ansatz wird dahingehend modifiziert, dass anstatt von *Concept- und Taskmodellen* die o.g. Anwendungsmodelle und Variantenbeschreibungen als Eingabe des Verfahrens dienen. Die weiteren Schritte werden entsprechend dem CAMELEON-Referenzframeworks umgesetzt und damit das technologieneutrale Anwendungsmodell in eine finale Benutzungsschnittstelle transformiert.

Das verwendete Verfahren ist so konzipiert, dass es als Eingabe lediglich das Anwendungsmodell benötigt und grenzt sich dadurch von den meisten bestehenden Ansätzen ab. Da es keine manuellen Schritte beinhaltet ist es für eine verteilte Nutzung geeignet, die insbesondere für die gemeinschaftliche Nutzung von Modellen benötigt wird. Durch die Anwendung des CAMELEON-Ansatzes ist das Vorgehen nicht neuartig, fügt ihm jedoch Verfahren zur Erstellung inhaltlicher Varianten und Kompositionen hinzu, die in bisherigen Umsetzungen nicht vorhanden sind.

Gemeinschaftliche Nutzung in einem Dienst-Ökosystem

Der dargestellte Lösungsansatz zur gemeinschaftlichen Nutzung setzt voraus, dass ein vollständig generierbares **Anwendungsmodell in einer universell interpretierbaren Form** vorliegt. Des weiteren setzt er voraus, dass die erfassten **Instanzdaten in einer universellen Form beschreiben werden**, um von beliebigen Dienst-Implementierungen genutzt werden zu können. Ist dies gegeben, können sowohl die Modelle als auch die funktionalen Elemente der dargestellten Lösungsarchitektur gemeinschaftlich genutzt werden.

Das Realisierungskonzept besteht in der Nutzung von **Ontologien** zur Repräsentation der Modelle des Lösungsansatzes. Die Aufgabe von Ontologien ist es, die Semantik von Entitäten, deren Eigenschaften und Zusammenhänge zu beschreiben (vgl. [95]). Sie eigenen sich daher zur Repräsentation des datenzentrierten Anwendungsmodells und erfasster Instanzdaten. Zur Beschreibung von Ontologien existieren allgemein akzeptierte Ansätze, wodurch sie sich als **universelle Repräsentation** eignen. Als Sprachen zur Wissensrepräsentation sind insbesondere RDF (Resource Description Framework) [44] und OWL (Web On-

tology Language) [94] verbreitet. Für diese existieren standardisierte Notationen (z.B. Turtle [179], OWL/XML[102], JSON-LD [213], Manchester [103]) und Werkzeuge zur Verarbeitung (z.B. Protégé [216]), SparQL, Programmbibliotheken [70, 161, 209]).

Die ontologische Beschreibung für Anwendungsmodelle lehnt sich an die Arbeiten von Gaulke et al. [76] und Khushraj et al. [115] an, wendet den Ansatz jedoch auf die Inhalte des datenzentrierten Modells an. Die Anwendungsmodelle werden mit Standard-Elemementen einer RDF/OWL-Ontologie beschrieben und damit eine **Applikations-Ontologie** erstellt, die hinsichtlich der enthaltenen Informationen äquivalent den Anwendungsmodellen ist.

Das Realisierungskonzept zur Bereitstellung der Instanzdaten für Dienste basiert ebenfalls auf ontologischen Beschreibungen. Hierzu werden die Inhalte und Struktur der von einem Dienst erwarteten Daten in Form einer **Dienst-Ontologie** beschrieben (vgl. Abschnitt 2.1.2). Da Anwendungsmodelle auf den dort spezifizierten Daten aufbauen, korrelieren die Elemente des Anwendungsmodells mit den Elementen in der Dienst-Ontologie und können automatisiert aufeinander abgebildet werden. Hierzu wird das Anwendungsmodell um **Korrelationsinformationen erweitert**, welche die Abbildung der Daten des Anwendungsmodells in die der Dienst-Ontologie beschreiben. Damit können zur Laufzeit erfasste Daten in eine **Dienst-Ontologie-Instanz** transformiert und weiterverarbeitet werden.

Durch die Nutzung ontologiebasierter Beschreibungen können auch die **funktionalen Elemente** der Lösungsarchitektur gemeinschaftlich nutzbar gemacht werden. Das Realisierungskonzept für die UI- und Instanzdaten-Transformation besteht darin, als Eingabe die o.g. Applikations-Ontologien zu nutzen und als Ergebnis Dienst-Ontologie-Instanzen zu erzeugen. Um die Verfahren darüber hinaus zum Aufbau eines Benutzungsschnittstellen-Ökosystems in einer verteilten Umgebung anbieten zu können, werden diese als Dienste mittels gängiger Web-Techologien (i.B. RESTful Webservices) umgesetzt.

Das Realisierungskonzept löst die in bisherigen Ansätzen offene Problemstellung der gemeinschaftlichen Nutzung. Durch die Verwendung von Ontologien sind sowohl Anwendungsmodelle als auch Eingabedaten in einer universellen Form beschrieben und von allen Teilnehmern mit gängigen Verfahren interpretierbar. Es entsteht eine gemeinsame Sprache, mit der Komponenten dezentral bereitgestellt und von gemeinschaftlich nutzbaren Verfahren verarbeitet werden können.

4. Methodisches Vorgehen

Dieses Kapitel stellt die Methodik dar, mit der die Arbeit durchgeführt und anhand der die Lösungsergebnisse abschließend evaluiert wurden.

Abschnitt 4.1 stellt das Forschungs-Paradigma und Vorgehensmodell zur Durchführung der Arbeit vor. Die Durchführung erfolgte nach dem *Design Science Research*-Ansatz [93], der einleitend vorgestellt wird. Im Weiteren wird dargestellt, welche konkreten Artefakte zur Beantwortung der Forschungsfragen in der Arbeit entwickelt werden. Dies bildet den Rahmen für die in den Kapiteln 5 - 9 beschriebene Umsetzung.

Abschnitt 4.2 stellt die Methodik zur Evaluation der Ergebnisse dar. Es werden etablierte Nachweismethoden in *Design Science Research* und deren Eignung zur Evaluation der Artefakte der Arbeit dargestellt. Abschließend werden die Bewertungskriterien aufgeführt und Methoden ausgewählt, anhand derer die Validierung und Evaluation durchgeführt werden. Dies bildet die Grundlage für die in Kapitel 10 dargestellten Nachweise.

4.1 Durchführung der Arbeit

Zur Beantwortung der Forschungsfragen (vgl. Abschnitt 1.3) wird in der Arbeit ein **konstruktivistischer Forschungsansatz** verfolgt (*Constructive Research*, *Design Science*)[52]. Klassische Forschungsansätze in den Naturwissenschaften fokussieren sich auf die Erklärung, Beschreibung, Untersuchung oder Vorhersage von Phänomenen in einem beobachtbaren Umfeld [6, 52, 77]. Während diese Ansätze das Problemumfeld vollständig erfassen und beschreiben wollen, zielt *Design Science* auf die explorative Erforschung neuartiger Probleme, in denen das Umfeld ggf. nicht vollständig bekannt ist [52, S. 13]. Der Ansatz erweitert den Forschungsfokus auf die konstruktive Lösung konkreter Problemstellungen und die Verbesserung einer existierenden Situation durch Modifikation bzw. Schaffung eines neuen Systems [6, 192, 229]. Die Grundlage bilden dabei durch Analyse gewonnene Erkenntnisse des Problembereichs, für welchen neuartige Artefakte entwickelt / konstruiert werden, die bisher nicht existieren [212]. Da das Ziel der Arbeit in der Schaffung eines praxistauglichen modellgetriebenen Ansatzes zur Erzeugung von Benutzungsschnittstellen besteht und dies eine Aufgabenstellung ist, die *per se* die Schaffung von Artefakten beinhaltet, eignet sich der *Design Science* Ansatz zu deren Durchführung.

Konkrete Ausprägungen von *Design Science* sind z.B. *Design Science Research* (DSR) [93] und *Action Design Research* [208]. Im Folgenden wird ein kurzer Überblick über den *Design Science Research*-Ansatz [93] und dessen Anwendung zur Durchführung dieser Arbeit gegeben. Eine umfassende Diskussion der Forschungsansätze und ausführliche Darstellung von DSR und dessen Durchführung findet sich in [52].

4.1.1 Design Science Research als Forschungs-Paradigma

Das Paradigma von *Design Science Research* wurde von Hevner et al. [93] zur Schaffung von neuem Wissen in einem *artifiziellen* [212] und neuartigen Umfeld (z.B. einem IT-System) vorgeschlagen.

> *"Design Science Research [...] seeks to create innovations [...] through which the analysis, design, implementation and use of information systems can be effectively and efficiently accomplished".* [93]

Hierbei werden Lösungen für konkrete Problemstellungen in Form bewertbarer *Artefakte* erstellt (z.B. Konstrukte/Konzepte, Modelle, Verfahren, Architekturen). Drescher et al. fassen die Ziele von DSR wie folgt zusammen:

> *"[DSR] is a methodological approach concerned with devising artifacts that serve human purposes. It is a form of scientific knowledge production that involves the* ***development of innovative constructions****, intended to solve* ***problems faced in the real world****, and simultaneously makes a kind of* ***prescriptive scientific contribution****. An important outcome of this type of research is an* ***artifact that solves a domain problem****, [...] which must be* ***assessed against criteria of value or utility****."* [52]

Aufgrund der Beziehung zu spezifischen Problemstellungen, erheben die resultierenden Lösungen nicht den Anspruch auf Vollständigkeit, sondern konzentrieren sich auf die Erfüllung der Anforderungen des konkret untersuchten Problems.

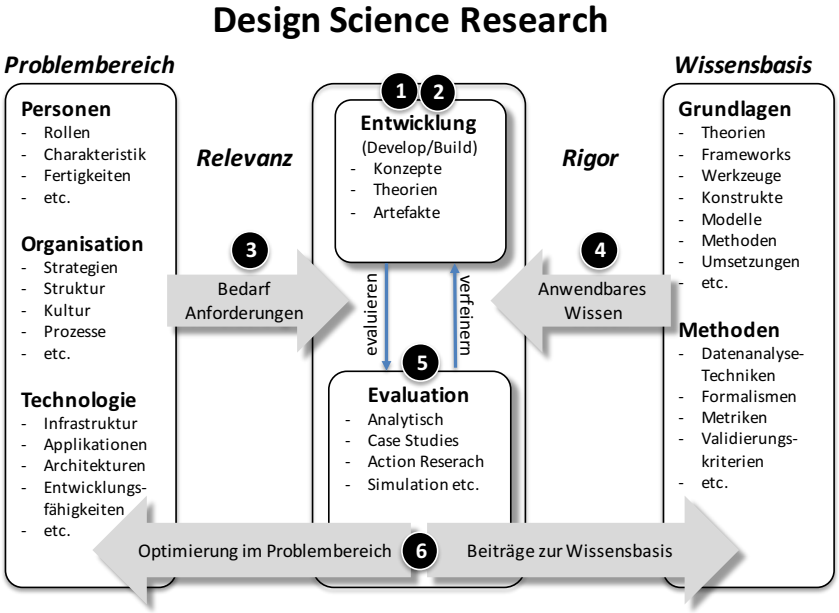


Abbildung 4.1. Design Science Research Elemente (Hevner et al.)

Abbildung 4.1 (mit Änderungen aus [93]) zeigt die Elemente von DSR und dient zur Illustration der Kernaspekte des Ansatzes. Hevner et al. definieren sieben Kriterien, die zu einer qualitativ hochwertigen Durchführung von DSR-Projekten zu beachten sind:

- **Design as Artifact**: Die Ergebnisse eines DSR-Prozesses sind neue Artefakte, die eine Problemstellung lösen ①. Eine Aufzählung möglicher Artefakttypen findet sich bei Dresch et al. [52], Kap. 5.2.
- **Design as Research Process**: Zur Konstruktion eines Artefakts ist es essentiell, dass Untersuchungen zum Verständnis des Problems mit wissenschaftlichen Methoden nachvollziehbar durchgeführt werden ②.

- **Problem Relevance**: Die Artefakte lösen Problemstellungen, die in einem konkreten *Problembereich* identifiziert werden. Es kann sich dabei um Verbesserungen eines Zustands oder der Lösung eines neuartigen Problems handeln. Der Problembereich bestimmt den Bedarf und die an die Lösung gestellten Anforderungen ③.
- **Research Rigor**: Die Durchführung basiert auf erprobten Methoden und berücksichtigt vorhandenes Wissen aus dem Problembereich (*Wissensbasis*) ④.
- **Design Evaluation**: Die Anwendbarkeit, Qualität und Wirksamkeit des Artefakts muss durch nachvollziehbare Evaluationsmethoden nachgewiesen werden ⑤. Insbesondere erfolgt die Entwicklung ①② und Evaluation ⑤ eines Artefakts in Iterationen, die das Ergebnis verfeinern.
- **Research Contribution**: Die Ergebnisse müssen verifizierbare Beiträge zur Forschung im Problembereich der erstellten Artefakte liefern ⑥ (Erweiterung der *Wissensbasis*, Verbesserungen im *Problembereich*)
- **Communication of Results**: Zur Nachvollziehbarkeit und Begutachtung des Ergebnisses müssen die Forschungsergebnisse kommuniziert werden.

Das DSR-Paradigma enthält jedoch kein Vorgehensmodell, nach dem Artefakte entwickelt werden. Es legt lediglich die Rahmenbedingungen und Kriterien zur Durchführung fest.

4.1.2 Vorgehensmodell zur Durchführung von DSR

In Dresch et al. [52] werden Vorgehensmodelle zur Durchführung von DSR-Projekten aufgeführt. Zur Erarbeitung der Artefakte schlagen Vaishnavi et al. [225] ein Modell vor, dass sich aus verbreiteten Vorgehensweisen in IT-Projekten ableitet und diese auf das DSR-Paradigma anwendet. Dieses wurde im Rahmen der Arbeit angewendet. Abbildung 4.2 stellt das Vorgehensmodell dar (mit Änderungen aus [225]).

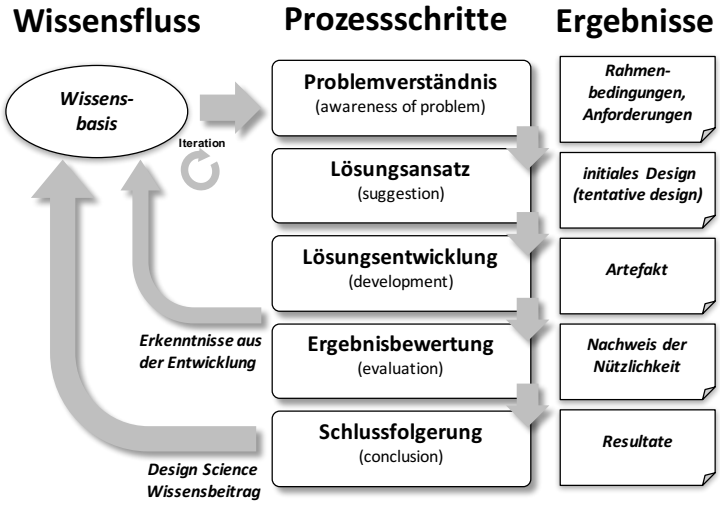


Abbildung 4.2. Vorgehensmodell zur DSR-Durchführung (Vaishnavi et al.)

Der erste Schritt dient in der Identifikation und Analyse des Forschungsproblems (*Problemverständnis*), welche als Ergebnis die Rahmenbedingungen und Anforderungen an die Lösung bestimmt. Daraus wird ein *Lösungsansatz* abgeleitet, der als Ergebnis ein initiales Design (*tentative design*) und Kriterien zur Bewertung des Resultats liefert. In weiteren, iterativ durchgeführten Schritten wird ein Artefakt entwickelt und hinsichtlich der Kriterien

evaluiert (*Lösungsentwicklung*, *Ergebnisbewertung*). Hierbei wird überprüft, ob das Ergebnis in der Umgebung standhält (*"fitness to adapt and survive within an environment"* [78]). Abweichungen vom erwarteten Ergebnis und neue Erkenntnisse erweitern die Wissensbasis und werden als Grundlage für eine Folge-Iteration herangezogen, die ggf. zur Anpassung des Lösungsansatzes und Änderung des Artefakts führt. Wird das Ergebnis für gut befunden, werden die Resultate als neuer *Design Science Wissensbeitrag* aufbereitet (*Reflection, Abstraction*) und tragen zur bestehenden Forschung bei. Eine detaillierte Beschreibung des Vorgehens kann [225] entnommen werden.

4.1.3 Anwendung auf die Arbeit

Zur Beantwortung der Forschungsfragen werden Artefakte i.S. des DSR-Paradigmas entwickelt, die in Abschnitt 3.3 bei der Vorstellung des Lösungsansatzes skizziert wurden. Zu deren Entwicklung wurde das dargestellte Vorgehensmodell entlang der in Tabelle 1.1 dargestellten Forschungsfragen und Themenbereiche angewendet.

Zur Schaffung des *Problemverständnisses* wurden der Problembereich und die Rahmenbedingungen analysiert und daraus Anforderungen abgeleitet. Diese waren die Grundlage zur Entwicklung des ersten Lösungsansatzes (*tentative Design*), aus welchem sich die Lösungsartefakte ergaben. Die Ergebnisse dieser Phase sind bereits in den Kapiteln 2 und 3 dargestellt. Die abgeleiteten Artefakte der Arbeit sind in Tabelle 4.1 mit (AF.0)-(AF.6) bezeichnet aufgeführt. Die Bezeichnungen werden im weiteren Verlauf als Referenzen verwendet.

Ref.	Artefakttyp	Themenbereiche und umsetzende Artefakte
Eigenschaften dialogbasierter Benutzungsschnittstellen (FF.1)		
(AF.0)*	Konzepte und Konstrukte	Eigenschaften und Anforderungen dialogbasierter Anwendungen Informationen zur Beschreibung der Eigenschaften
Modellierung von Benutzungsschnittstellen und Varianten (FF.2)		
(AF.1)	Modell	Metamodell und Notation zur Modellierung dialogbasierter Anwendungen (Anwendungsmodell)
	Modell	Metamodell und Notation zur Modellierung fachlicher Varianten (Variantenmodell)
Komposition von Benutzungsschnittstellen (FF.3)		
(AF.2)	Modell	Erweiterung des Anwendungsmodells zur Erstellung von Kompositionen
	Modell	Nutzung des Varaintenmodells zur Anpassung von Komponenten
Generierung von Benutzungsschnittstellen, Varianten und Kompositionen (FF.4)		
(AF.3)	Verfahren	Verfahren zur Herleitung finaler Benutzungsschnittstellen aus den Modell-Artefakten
Gemeinschaftliche Nutzung in Dienst-Ökosystemen (FF.5)		
(AF.4)	Modell	Gemeinschaftlich nutzbare Repräsentation des Anwendungsmodells (Universelle Modellierung)
(AF.5)	Modell	Modellierung der Anbindung von Benutzungsschnittstellen an Dienste
(AF.6)	Architektur	Infrastruktur zur Nutzung in einem Dienst-Ökosystem

* = informelles Artefakt ohne technische Evaluation.

Tabelle 4.1. Verwendete Artefakttypen und Artefakte

Die Umsetzung der Artefakte erfolgte aufeinander aufbauend und iterativ mit Bezug zu den Forschungsfragen (*Lösungsentwicklung*). Die Darstellung der Ergebnisse erfolgt in den Kapiteln 5-9.

Die Untersuchung von Forschungsfrage (FF.1) zielt auf eine *Vertiefung des Problemverständnisses*. Das Ergebnis ist ein informelles Artefakt (AF.0) das Konzepte und vorgefundene Konstrukte des Problembereichs beschreibt. Es dient dem Verständnis der Forschungsfragen (FF.2)-(FF.5) und resultiert in konkreten technischen Anforderungen an die weiteren Artefakte. Zur Beantwortung der Forschungsfragen (FF.2)-(FF.5) wurden aufeinander aufbauend die technischen Artefakte (AF.1)-(AF.6) konzipiert, entwickelt und umgesetzt (*Lösungsentwicklung*). Die Artefakte wurden anhand in DSR-Projekten verbreiteter Methoden (s. Abschnitt 4.2) hinsichtlich ihrer technischen Machbarkeit, Anwendbarkeit auf das Problem und Nutzbarkeit in der Umgebung evaluiert (*Ergebnisbewertung*). Die Resultate der Evaluation ergaben ggf. neue Erkenntnisse und weitere Anforderungen, die in die nächste Iteration einflossen.

Die Kommunikation der Ergebnisse (*Schlussfolgerung*) erfolgte im Verlauf der Arbeit im Rahmen von Veröffentlichungen als Konferenzbeiträge und Publikationen.

4.2 Evaluation der Ergebnisse

Design Science Research als Forschungs-Paradigma sieht als integralen Bestandteil die Evaluation vor, beinhaltet jedoch keine feste Menge an Methoden zu deren Durchführung. Im Umfeld haben sich Nachweismethoden etabliert, auf die im Rahmen der Arbeit zurückgegriffen wird.

4.2.1 Nachweismethoden in Design Science Research

Hevner et al. beschreiben in [93] einige Methoden zur Evaluierung im Rahmen von DSR erstellter Artefakte. Sie geben jedoch keine Hinweise, welche Methoden sich für konkrete Artefakttypen eignen. Die dort genannten Evaluierungen finden zudem erst nach Implementierung der Artefakte statt und fokussieren sich eher auf die Umsetzung des Lösungsansatzes als auf das Lösungskonzept. Darauf aufbauende Veröffentlichungen schlagen daher ergänzend vor, bereits vor der Umsetzung Konzepte, Designansätze etc. zu evaluieren. Das Ziel ist es, bereits in frühen Phasen geeignete Lösungsoptionen auswählen zu können [122].

Evaluierungsarten: *ex-ante* und *ex-post*. Pries-Heje et al. [178] schlagen eine Kombination von Evaluierungen zu unterschiedlichen Zeitpunkten im DSR-Prozess vor. Hierbei wird zwischen Evaluierungen vor bzw. während der Erstellung des Artefakts (**ex-ante**) und nach dessen Erstellung (**ex-post**) unterschieden. Zudem findet eine kontinuierliche Evaluierung **während der Lösungsfindung** und Artefaktumsetzung in Iterationen statt. *Ex-ante*-Untersuchungen dienen der Evaluierung der Konzepte, *ex-post*-Untersuchungen die deren Umsetzung.

Abbildung 4.3, links (mit Änderungen aus [178]) zeigt eine Gliederung des DSR-Vorgehens in eine *Research* und eine *Construction*-Phase. Während der *Research*-Phase werden vorwiegend *ex ante*-Evaluierungen vorgenommen, in welchen potentielle Lösungen in Bezug auf die Anforderungen evaluiert werden.

Basierend auf den Ergebnissen der *ex-ante*-Validierungen erfolgt die Auswahl und *Konstruktion* (z.B. Implementierung) einer Lösung, die *ex post*-Untersuchungen unterzogen wird. Dies soll die Machbarkeit und Nutzbarkeit erstellter Artefakte nachweisen.

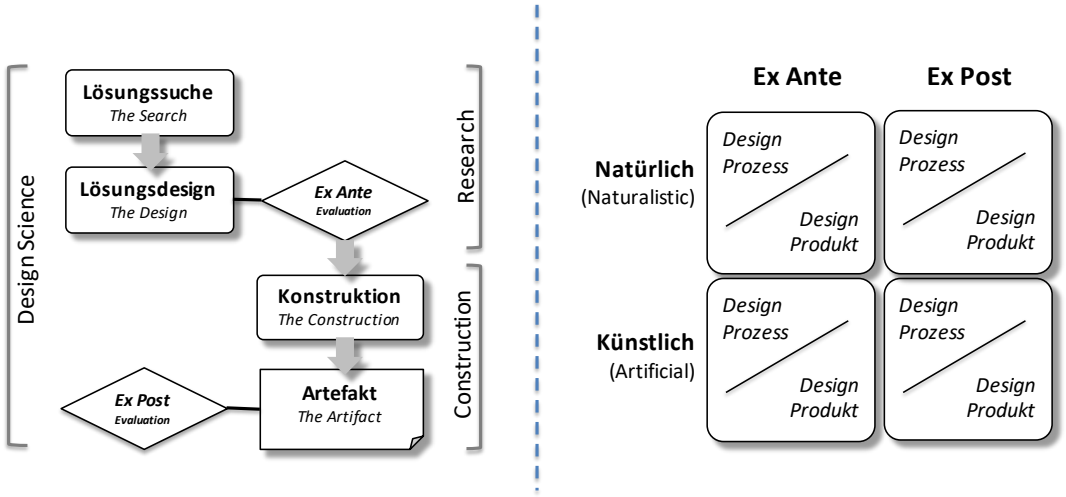


Abbildung 4.3. *ex-ante*- und *ex-post*-Evaluierung (Pries-Heje et al.)

Pries-Heje et al. unterscheiden dabei zwischen künstlichen (*artificial*) und realen (*naturalistic*) Szenarien, in welchen Evaluierungen durchgeführt werden können (Abbildung 4.3, rechts). Evaluierungen im einem **künstlichen Szenario** sind dabei Studien in einem konstruierten Umfeld. Hierbei wird validiert, dass eine Annahme in einem kontrollierten, dennoch *realitätsnahen Umfeld* erfüllt wird. In einem **natürlichen Szenario** durchgeführte Evaluierungen hingegen untersuchen die realen Auswirkungen einer Lösung. Jedes Szenario bietet eigene Vorteile: das künstliche Szenario bietet größere Kontrolle über die Einflussfaktoren einer Untersuchung, wohingegen ein natürliches Szenario Ergebnisse für einen konkreten Anwendungsfall liefert [178].

Evaluierungsmethoden in DSR-Projekten. In Tabelle 4.2 sind Methoden zusammengetragen, die *ex-ante*/*ex-post* und in *künstlichen*/*realen Szenarien* verwendet werden können. Die Tabelle folgt dabei der Darstellung in Hevner et al. [93] und beschreibt die Zielsetzung der Methoden. Die Tabelle wurde ergänzt um weitere Methoden aus Peffers et al.. Diese wurden im Rahmen einer Analyse durchgeführter DSR-Projekte identifiziert [165].

In Ergänzung zu Hevner et al. geben Peffers et al. Hinweise darauf, welche Evaluierungsmethoden sich für bestimmte Artefakttypen eignen. Es wurde untersucht, welche Methoden in existierenden DSR-Projekten vorwiegend für welche Artefakte verwendet wurden. Die Untersuchungen liefern *best practices* und damit einen Indikator für die Auswahl von Evaluierungsmethoden für eigene DSR-Projekte.

4.2.2 Bewertung und Auswahl geeigneter Nachweismethoden

Die Untersuchungen in Peffers et al. zeigen, dass ein verbreitetes Vorgehen in DSR-Projekten die **prototypische Implementierung** und anschließende Evaluation über **technische Experimente** und **Fallstudien** ist [165]. In besonderem Maße gilt dies, wenn es sich bei den betrachteten Artefakten um **Modelle** und **Verfahren** handelt. Da die Ergebnisse dieser Arbeit konkrete Komponenten zur Erzeugung einer Benutzungsschnittstelle darstellen und die Bewertungskriterien (vgl. Abschnitt 4.2.3) vorwiegend technischen und vergleichenden Charakter haben, sind zur Evaluierung insbesondere die in Tabelle 4.2 unter den Abschnitten

Evaluierungs-methoden		Beschreibung
1. Observational		
Case Study	1,2	1: Study artifact in depth in business environment. 2: Application of an artifact to a real-world situation, evaluating its effect on the real-world situation.
Field Study	1	Monitor use of artifact in multiple projects
Action Research	2	Use of an artifact in a real-world situation as part of a research intervention, evaluating its effect on the real-world situation.
Expert Evaluation	2	Assessment of an artifact by one or more experts
Subject-based Experiment	2	A test involving subjects to evaluate whether an assertion is true.
2. Analytical		
Static Analysis	1	Examine structure of artifact for static qualities (e.g., complexity)
Architecture Analysis	1	Study fit of artifact into technical IS architecture
Optimization	1	Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
3. Experimental		
Prototype	2	Implementation of an artifact aimed at demonstrating the utility or suitability of the artifact.
Controlled Experiment	1	Study artifact in controlled environment for qualities (e.g., usability)
Technical Experiment	2	A performance evaluation of an algorithm implementation using real-world data, synthetic data, or no data, designed to evaluate the technical performance, rather than its performance in relation to the real world.
Simulation	1	Execute artifact with artificial data
4. Testing		
Functional (Black Box) Testing	1	Execute artifact interfaces to discover failures and identify defects
Structural (White Box) Testing	1	Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive		
Informed Argument/ Logical Argument	1,2	1: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility 2: An argument with face validity. [Face validity is the extent to which a test is subjectively viewed as covering the concept it purports to measure]
Illustrative Scenario	1,2	Application of an artifact to a synthetic or real-world situation aimed at illustrating suitability or utility of the artifact.

***1 = Hevner et al.**

***2 = Peffers et al**

Tabelle 4.2. DSR-Evaluierungsmethoden (Hevner et al., Peffers et al.)

Observational, *Experimental* und *Descriptive* aufgeführten Methoden sinnvoll anwendbar. Insbesondere sind dies die Methoden

- **Prototype** als prototypische Implementierung eines Artefakts und damit Nachweis der Machbarkeit
- **Illustrative Scenario** als Umsetzung eines fokussierten Anwendungsfalls als Nachweis der Anwendbarkeit
- **Case Study** als realitätsnahe Umsetzung eines konkreten Anwendungsfalls zum Nachweis der Nutzbarkeit
- **Action Research** als Nachweis der Nutzbarkeit in einem existierenden Umfeld
- **Expert Evaluation** als Mittel zur Bewertung der Qualität der Ergebnisse

Diese Methoden setzen die Artefakte konkret um (*Prototype*) und verwenden sie in variierenden Konstellationen (*Case Study, Action Research, Illustrative Scenario, Action Research*)

anhand derer die Evaluationskriterien überprüft werden können (z.B. Vergleich mit bestehenden Anwendungen, Nachweis geforderter Eigenschaften und Funktionalität).

Evaluierungen durch Experten (*Expert Evaluations*) dienen der Prüfung der Ergebnisqualität. Sie werden in Kopenhagen et al. [122] als aufwendig in der Durchführung eingestuft und daher insbesondere für größere Projektsettings vorgeschlagen. Sie können sowohl *ex-ante*, als auch *ex-post* zur Verbesserung der Ergebnisse beitragen. Im Verlauf der Arbeit erfolgten Experten-Evaluierungen unstrukturiert, um den Aufwand zu minimieren, aber dennoch die damit verbundene Qualitätssicherung zu erhalten. In wenigen Fällen kann auch eine logische Argumentation (**Logical Argument**) technische oder empirische Untersuchungen ersetzen - sofern es sich um offensichtlich nachvollziehbare Aspekte handelt.

Die in Tabelle 4.2 unter **Analytical** und **Testing** aufgeführten Methoden sind zum Nachweis der Thesen der Arbeit hingegen wenig geeignet. Sie suchen insbesondere die Qualität des erstellten Artefakts zu messen und liefern Aussagen zur technischen Qualität einer Umsetzung. Dasselbe gilt für technische Experimente (**Technical Experiment**), z.B. zum Nachweis der Stabilität und Performanz. Da im Rahmen der Arbeit eine prototypische Implementierung erfolgt, sind Evaluierungen zum Verhalten zu diesem Zeitpunkt noch wenig aussagekräftig und erst sinnvoll, wenn die grundsätzliche Funktion nachgewiesen wurde.

Als wenig sinnvoll werden Evaluationen erachtet, die auf **empirischen Untersuchungen** basieren und eher subjektiv-qualitativer Natur sind. So finden sich in DSR-Projekten häufig Fallstudien mit Nutzergruppen (Tabelle 4.2, **Subject-based Experiments**, **Controlled Experiments**), welche z.B. die Einfachheit eines Ansatzes nachzuweisen suchen. Problematisch ist hierbei, dass diese Untersuchungen mit hohen Nutzerzahlen erfolgen müssen, um aussagekräftige Ergebnisse zu erhalten. Im vorliegenden Fall besteht die besondere Herausforderung darin, dass die Nutzergruppe einen hohen technischen und einheitlichen Kenntnisstand zur Erstellung von Benutzungsschnittstellen für unterschiedliche Technologien besitzen muss. Ein Vergleich mit bestehenden Ansätzen ist zudem nur unzureichend möglich, da es keine vergleichbare Lösung gibt. Eine solche Untersuchung verspricht daher Ergebnisse mit lediglich geringer Aussagekraft.

Ebenfalls mit empirischen Mitteln könnte die Qualität der erzeugten Benutzungsschnittstellen untersucht werden, z.B. nach Nutzbarkeits-Gesichtspunkten (*Usability*). Dies kann durch Feldstudien (**Field Study**) in der Breite erfolgen. Für die Bewertung des vorgeschlagenen Ansatzes und der Thesen ist dies aktuell nur begrenzt sinnvoll, da sich die Arbeit aufgrund des Umfangs auf ausgewählte Anwendungsfelder beschränken muss. So würde eine Untersuchung keine Aussage über die Qualität und Leistungsfähigkeit des Ansatzes selbst liefern, sondern lediglich für ausgewählte Endprodukte, die zudem prototypischer Natur sind. Eine breite Feldstudie ist zu einem späteren Zeitpunk jedoch sinnvoll.

4.2.3 Anwendung auf die Arbeit

Im Verlauf der Arbeit werden Konzepte, Modelle, Verfahren und Architekturmuster als Artefakte entwickelt, die eine vollständig automatisierten Erzeugung von Varianten und Kompositionen sowie eine gemeinschaftlich Nutzung von Benutzungsschnittstellen in verteilten Dienst-Ökosystemen ermöglichen. Diese Artefakte wurden in Abschnitt 4.1.3, Tabelle 4.1 mit (AF.0)-(AF.6) aufgeführt und sind Gegenstand der Evaluation.

Die Artefakte werden mit dem Ziel entwickelt, die in der Problemstellung dargestellten Herausforderungen zu lösen und hierzu in Abschnitt 1.2 Eigenschaften dargestellt, die eine Lösung aufweisen muss. Daraus leiten sich die **Kriterien zur Bewertung** der erstellten Artefakte für die Evaluation ab. Mit der Nutzung der Artefakte müssen folgende Kriterien erfüllt werden:

- **(K.1)** Erzeugung von Benutzungsschnittstellen, die funktional manuell erstellten entsprechen
- **(K.2)** Erzeugung inhaltlicher Varianten
- **(K.3)** Komposition bestehender Benutzungsschnittstellen
- **(K.4)** Vollständig automatisierte Erzeugung technischer Varianten
- **(K.5)** Beschreibung gemeinschaftlich nutzbarer Benutzungsschnittstellen, die als Komponenten in Anwendungen Dritter integrierbar sind
- **(K.6)** Erzeugung von Benutzungsschnittstellen, die an beliebige Dienste eines Dienst-Ökosystems angebunden werden können

Tabelle 4.3 zeigt die gewählten Methoden, die zum Nachweis der Erfüllung angewendet werden. Für jedes der erstellten technischen Artefakte (AF.1)-(AF.6) sind neben dessen Typ die Kriterien aufgeführt, die vom Artefakt adressiert und damit validiert bzw. evaluiert werden. Zu jedem Artefakt ist zudem aufgeführt, welche *ex-ante*- und *ex-post*- Untersuchungen angewendet werden. Das in Abschnitt 4.1.3 beschriebene Artefakt (A.0) als informelles Artefakt stellt einen Sonderfall dar. Da es Analyseergebnisse beinhaltet, ist eine technische Evaluation nicht möglich.

Artefakt		Typ	Kriterien	Methoden / Verfahren		Nachweis
				ex ante	*ex post*	
Modellierung und automatische Erzeugung von Benutzungsschnittstellenvarianten						
(AF.1)	**Metamodell für Anwendungen und Varianten**	Modell	**K.1** **K.2**	Experten-Evaluation (A.1)	Prototyp Szenario	Machbarkeit Anwendbarkeit
(AF.2)	**Modellierung von Kompositionen**	Modell	**K.3**	Experten-Evaluation (A.2)	Prototyp Szenario	Machbarkeit Anwendbarkeit
(AF.3)	**Verfahren zur Generierung von Benutzungsschnittstellen**	Verfahren	**K.4**	Experten-Evaluation (A.3)	Prototyp Szenario Case Study Action Research	Machbarkeit Anwendbarkeit Nutzbarkeit
Gemeinschaftliche Nutzung in einem Dienst-Ökosystem						
(AF.4)	**Universelle Modellierung**	Modell	**K.5**	Experten-Evaluation (A4.1)	Prototyp Szenario	Machbarkeit Anwendbarkeit
(AF.5)	**Modellierung der Dienstanbindung**	Modell	**K.6**	Experten-Evaluation (A4.2)	Prototyp Szenario Case Study	Machbarkeit Anwendbarkeit Nutzbarkeit
(AF.6)	**Infrastruktur für Shared UIs**	Architektur	**K.6**	Experten-Evaluation (A4.3)	Prototyp Szenario Case Study Action Research	Machbarkeit Anwendbarkeit Nutzbarkeit

Tabelle 4.3. Artefakte und Evaluierungsmethoden

***ex-ante* Untersuchungen.** Für alle Artefakte wurde vor deren Konstruktion ein Abgleich mit den Anforderungen vorgenommen, die für das Artefakt in Tabelle 3.1 formuliert bzw. sich aus den Analyseergebnissen des Artefakts (AF.0) ergaben. Dies erfolgte in Form einer Experten-Evaluation vor der Umsetzung des entsprechenden Artefakts in jedem Iterationsschritt. Dieser Abgleich diente der Verifikation der Vollständigkeit bei der Umsetzung der gefundenen Eigenschaften der Modelle und Verfahren.

Im Rahmen der Arbeit wird der *ex-ante*-Abgleich mit den Anforderungen **durchgängig in den Ergebnisdiskussionen der Umsetzungskapitel** dargestellt, auf die an dieser Stelle verwiesen wird (Abschnitte 5.7, 6.6, 7.5, 8.6, 9.7).

***ex-post* Untersuchungen.** Die *ex-post*-Untersuchungen umfassen die Nachweise für Machbarkeit, Anwendbarkeit und Nutzbarkeit des Ansatzes. In Tabelle 4.3 sind in der Spalte *Methode/Verfahren* unter *ex-post* die hierzu gewählten Verfahren aufgeführt.

Zum Nachweis der **Machbarkeit** wird die prototypische Umsetzung gewählt. Für jedes Artefakt (Modell, Verfahren, Architektur) wird hierzu eine Implementierung erstellt, welche die in den Kapiteln 6 bis 9 beschriebenen Modelle und Verfahren technisch umsetzt.

Die **Anwendbarkeit** der Artefakte wird über deren praktische Nutzung zur Beschreibung und Generierung von Anwendungsvarianten validiert. Hierzu werden *künstliche Szenarien* verwendet, an denen die Erfüllung der Anforderungen an den Lösungsansatz gezeigt werden. Sofern dies möglich ist, werden hierzu existierende Benutzungsschnittstellen zum Vergleich herangezogen und untersucht.

Die Untersuchung zur **Nutzbarkeit** des Ansatzes erfolgt sowohl als artefaktübergreifende Evaluation in Form von Fallstudien (*Case Studies*), als auch durch Nachweis der Nutzbarkeit von Ergebnissen in einem realen Umfeld (*Action Research*). In den *Case Studies* werden die prototypisch implementierten Ergebnisse in realitätsnahen Anwendungsfällen über Fallstudien evaluiert. Darüber hinaus wird in den *Action Research*-Umsetzungen der Einsatz der Ergebnisse in realen Anwendungen gezeigt. Die Zusammenarbeit mit der Allianz Deutschland AG ermöglichte hierbei den Nachweis der Nutzbarkeit und konkrete Verbesserungen in Anwendungen, die sich heute im produktiven Einsatz der Vermittler- und Kundenportale befinden.

Die *ex-post*-Untersuchungen der Arbeitsergebnisse werden in Kapitel 10 durchgeführt. Dort wird einleitend beschrieben, welche konkreten Untersuchungen erfolgten und deren Durchführung dargestellt.

Begleitend zu den in Tabelle 4.3 aufgeführten Methoden wurden **Experten-Gespräche** mit Architekten und Entwicklern der Allianz Deutschland durchgeführt. Sie dienten *ex-ante* zur Verifikation der Anforderungen und *ex-post* der subjektiven Bewertung der erzeugten Anwendungen. Die *ex-post* durchgeführten Gespräche erfolgten insbesondere bei den als *Action Research* durchgeführten Evaluationen, da es sich dabei um Anwendungen aus dem realen Umfeld der Allianz handelt, die in den produktiven Betrieb überführt wurden. Die Experten-Gespräche erfolgten jedoch nicht formal (z.B. über Fragebögen) und wurden als Bestandteil der aufgeführten Untersuchungen durchgeführt. Sie sind daher nicht gesondert in der Tabelle aufgeführt.

5. Analyse der Eigenschaften dialogbasierter Anwendungen

Dieses Kapitel stellt die Eigenschaften von Benutzungsschnittstellen dialogbasierter Anwendungen und daraus resultierende Anforderungen an deren Modellierung vor. Hierzu werden die zur umfassenden Generierung dialogbasierter Benutzungsschnittstellen und deren technischer und inhaltlicher Varianten erforderliche Informationen dargestellt und als Ergebnis die Anforderungen an den Informationsgehalt eines Modells abgeleitet. Diese bilden die Grundlage für den Modellierungsansatz.

Beitrag des Kapitels. Nach meinem besten Wissen existiert in der Literatur keine systematische Analyse der Eigenschaften dialogbasierter Anwendungen, welche als Grundlage zur technologieneutralen Modellierung und umfassenden Generierung von Benutzungsschnittstellen dienen kann. Der Beitrag dieses Kapitels besteht in der Spezifikation der Informationen, die zur Generierung dialogbasierter Benutzungsschnittstellen benötigt werden und der daraus resultierenden Anforderungen an die Modellierung.

5.1 Zielsetzung und Vorgehen

Im in Kapitel 3 dargestellten Lösungsansatz wird ein Anwendungsmodell als zentrales Artefakt zur Beschreibung von dialogbasierten Benutzungsschnittstellen und deren Varianten vorgeschlagen. Dies setzt voraus, dass alle zur Generierung einer nicht-trivialen Benutzungsschnittstelle benötigten Informationen in einem solchen Anwendungsmodell enthalten sind. Daraus ergeben sich die in Abschnitt 1.3 unter **(FF.1)** zusammengefassten Forschungsfragen:

- Welche Informationen werden zur Generierung einer nicht-trivialen Benutzungsschnittstelle benötigt?
- Welche Informationen sind zur Herleitung inhaltlicher und technischer Varianten erforderlich?
- Können diese Informationen technologieneutral formuliert werden, sodass sie auf multiple Interaktions- und Technologiekontexte anwendbar sind?

Das Ziel des Kapitels liegt in der Identifikation dieser Informationen sowie daraus resultierender Anforderungen an die Modellierung.

Als Vorgehen wurde ein top-down-Ansatz gewählt, der auf der Analyse der Eigenschaften bestehender und in der Praxis genutzter Anwendungen basiert. Daraus wurden Struktur- und Verhaltensmuster abgeleitet, welche in Anforderungen mündeten.

Datengrundlage. In Zusammenarbeit mit der Allianz Deutschland AG wurden bestehende dialogbasierte Formularanwendungen aus dem Kunden- und Vertriebsumfeld ausgewählt und untersucht. Bei der Auswahl und der Diskussion der Beobachtungen und Ergebnisse waren Architekten und Entwickler aus verbundenen IT-Bereichen beteiligt, die zur Validierung des praktischen Nutzens und der Vollständigkeit beitrugen. Aufgrund der hohen Variabilität und Vielfalt an Technologien im Umfeld von Benutzungsschnittstellen ist Vollständigkeit nicht abschließend nachweisbar. Die Einschätzung wurde daher nach bestem Wissen von Experten im Rahmen ihres fachlichen Umfelds vorgenommen und berücksichtigte dort typische und häufig vorkommende Muster.

Die Auswahl von Anwendungen erfolgte unter der Prämisse, dass diese ein möglichst weites qualitatives Spektrum für die Analyse abbilden. Insbesondere sollen sie sich hinsichtlich der Komplexität der Abläufe, der Struktur als auch der Dynamik der Benutzungsschnittstellen unterscheiden. In Anhang A.1 wird die Datengrundlage und Auswahl der untersuchten Anwendungen im Detail erläutert und hier lediglich zusammengefasst:

- Für die Analyse wurden als Prozessbereiche **Kunden- und Vermittler- Anwendungen** als besonders geeignet identifiziert, da hier vorwiegend dialogbasierte Benutzungsschnittstellen eingesetzt werden. Zudem überdecken sich in diesen Bereichen viele Prozesse, die nutzergruppenspezifisch in Varianten angeboten werden.

- Als analyserelevante Anwendungstypen wurden **Self-Service-Anwendungen, Antragstrecken und Dialoge zur Erfassung von Risikofragen** identifiziert. Hierbei liefern Self-Service-Anwendungen grundlegende Anforderungen an die Benutzungsschnittstelle. Antragstrecken fügen Erkenntnisse zu komplexen Dialogabläufen aufgrund bereits erfasster Daten und Geschäftslogik hinzu. Risiko-Dialoge ergänzen Erkenntnisse im Umgang mit Backend-Systemen und der Integration in andere Anwendungen.

- Der Fokus liegt auf Anwendungen mit **grafischen Benutzungsschnittstellen**, da diese Interaktionsform zum Zeitpunkt der Analyse am häufigsten vertreten ist und damit hohe Praxisrelevanz besitzt. Die betrachteten Technologien ergaben sich durch die Plattformen, auf denen die untersuchten Anwendungen bereitgestellt werden (Internet, Extranet und Desktop-Geräte der Vermittler).

Die Grundlage für die **Analyse der Kunden-Anwendungen** bilden die Anwendungen im Kundenportal der Allianz Deutschland AG[31]. Hierbei wurden Anwendungen im *öffentlich zugänglichen Bereich* (*http://allianz.de*), dem Kunden vorbehaltenen und passwortgeschützten *geschlossenen Bereich* (*http://meine.allianz.de*) sowie *Kunden-Self-Services* (*http://www.allianz.de/service/*) getrennt ausgewertet. In Summe wurden **203 Kunden-Anwendungen** betrachtet.

Im **öffentlich zugänglichen Bereich** wurden **42 Anwendungen** identifiziert, von denen 93% als dialogbasiert i.S. der Arbeit eingestuft wurden. Im **angemeldeten Bereich** wurden **21 Anwendungen** betrachtet, von denen 67% als dialogbasiert eingestuft wurden. Die Komplexität der dialogbasierten Anwendungen war dabei moderat (64%) bis anspruchsvoll (29%). Hinzu kommen in beiden Bereichen **Kunden-Selfservices** wie Kontaktaufnahme-Formulare (aktuell **140 sog. *Anliegen***), die zu 100% dialogbasiert arbeiten und überwiegend moderate, selten anspruchsvolle Komplexität aufweisen (s. Details in Anhang A.1).

Als Grundlage für die **Analyse von Vermittler-Anwendungen** wurden die dialogbasierten Anwendungen des Vermittler-Desktops (Arbeitsplatzrechner, Laptop) herangezogen. Dabei handelt es sich um ca. **79 Bausteine** für die fachlichen Themen *Versicherung (43)*, *Vorsorge (24)* und *Vermögen (4)* sowie *Broschüre-Anwendungen (8)* die als Bausteine in eine Rahmenanwendung integriert werden. Davon sind **66% Antragstrecken** und **20% Beratungs-**

[31] Neben dem Allianz-Portal wurden bei der quantitativen Analyse anfangs weitere Versicherungs- und Finanzdienstleister-Portale einbezogen (z.B. Deutsche Kredit Bank, Deutsche-Bank, Techniker Krankenkasse, Barmer Ersatzkasse; vgl. Anhang A.1). Die Anwendungen sowie die Verhältnisse waren hier sehr ähnlich. Daher wurde die weitere Analyse auf die Allianz-Anwendungen beschränkt.

anwendungen, die sich auf einem **überwiegend anspruchsvollen Komplexitätsniveau** befinden (s. Details in Anhang A.1).

Die Anwendungen wurden aufgrund ihrer Komplexität und Ähnlichkeit geclustert und in Komplexitätsklassen eingeteilt. Aus den Gruppen wurden repräsentative Vertreter ausgewählt, die in der weiteren Analyse detailliert untersucht wurden.

Vorgehensweise. Abbildung 5.1 zeigt die Schritte, die zur Analyse durchlaufen wurden. Im ersten Schritt wurde identifiziert, welche **strukturellen und verhaltensrelevanten Eigenschaften** dialogbasierte Benutzungsschnittstellen aufweisen und wie sich diese ggf. bei Varianten unterscheiden. Es wurde betrachtet,

- ob ein einheitliches Architekturmuster zugrunde gelegt werden kann
- welchen strukturellen Merkmale die Benutzungsschnittstellen aufweisen
- welche Navigationsformen verwendet werden
- mit welchen Interaktionselementen die Ein-/Ausgabe der Daten erfolgt
- wie sich die Benutzungsschnittstellen während der Eingabe verhalten

Zur Ergänzung und Überprüfung der vorgefundenen Muster wurde Literatur zu den Themenbereichen *Benutzungsschnittstellen-Design* (z.B. [35, 205, 210, 224]), *Mensch-Computer-Interaktion* (z.B. [48, 89]) und zu *Interaktionsmustern* (z.B. [49, 153, 222, 231]) hinzugezogen.

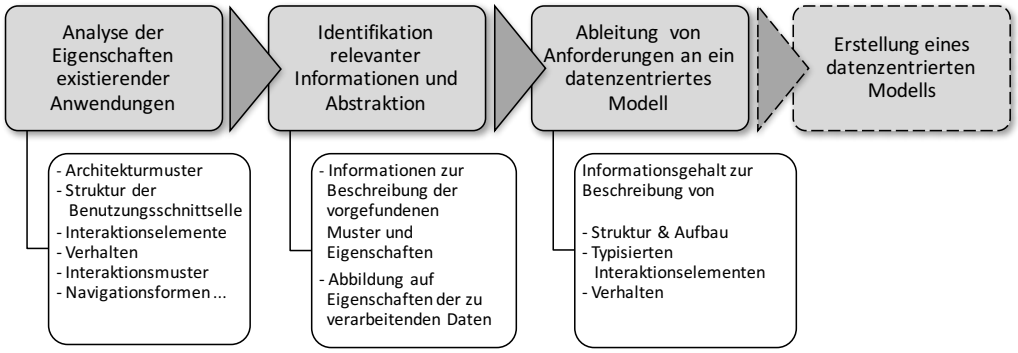


Abbildung 5.1. Vorgehensweise zur Analyse der Eigenschaften dialogbasierter Anwendungen und deren Varianten

In nächsten Schritt wurde untersucht, welche Informationen zur Beschreibung von Benutzungsschnittstellen mit den vorgefundenen Eigenschaften benötigt werden. Insbesondere wurde betrachtet, ob die vorgefundenen Muster technologieneutral erfolgen und Struktur, Aussehen und Verhalten aus Eigenschaften der Daten abgeleitet werden kann. Abschließend wurden daraus Anforderungen an die Inhalte eines Anwendungsmodells abgeleitet. Die resultierenden Anforderungen sind die Grundlage für den Aufbau des datenzentrierten Anwendungsmodells in Kapitel 6.

Die folgenden Abschnitte beschreiben die Ergebnisse im Detail. In **Abschnitt 5.2** wird das aus der Analyse abgeleitete Architekturmuster dargestellt, das im weiteren Verlauf der Arbeit als konzeptioneller Rahmen verwendet wird. Die folgenden **Abschnitte 5.3-5.6** beschreiben die funktionalen Eigenschaften von Benutzungsschnittstellen entlang der Themenbereiche *Struktur und Aufbau* (Abschnitt 5.3), *Typisierte Ein-/Ausgabe von Daten* (Abschnitt 5.4) und deren *Verhalten* (Abschnitt 5.5). Diese werden in **Abschnitt 5.6** um nicht-funktionale Eigenschaften ergänzt. Für jeden Themenbereich werden zum einen die grundlegenden Eigen-

schaften, zum anderen die Unterschiede in inhaltlichen und technischen Varianten dargestellt und abschließend die Anforderungen an die Modellierung abgeleitet. **Abschnitt 5.7** fasst die Ergebnisse zusammen und diskutiert sie hinsichtlich der Ziele.

5.2 Architekturmuster dialogbasierter Anwendungen

Dialogbasierte Formularanwendungen sammeln in einer Folge von Schritten Daten vom Nutzer. Während der Verwendung der Anwendung wird Programmlogik ausgeführt, die das Verhalten der Anwendung beeinflusst. Dieses Verhalten basiert auf den bereits erfassten Daten (vgl. Abschnitt 2.2.2). Zur Umsetzung dieser Eigenschaften benötigen Anwendungen zur Laufzeit folgende Komponenten:

- **Benutzungsschnittstelle** zur Interaktion
- **Datenhaltung** zur Verwaltung des Zustands
- **Operationen** zur Umsetzung des Verhaltens

Diese Komponentenaufteilung findet sich häufig in Designansätzen zum Aufbau von Anwendungen. Prominente Beispiele sind das von Trygve Reenskaug 1979 entwickelte *Model-View-Controller*-Muster (MVC) [24, 190] oder das von Microsoft vorgeschlagene *Model-View-Viewmodel*-Muster (MVVM) [75]. Diese Ansätze fokussieren auf Anwendungen mit grafischen Benutzungsschnittstellen, können jedoch generalisiert auf dialogbasierte Anwendungen angewendet werden. Abbildung 5.2 zeigt das Zusammenspiel der Komponenten des Architekturmusters.

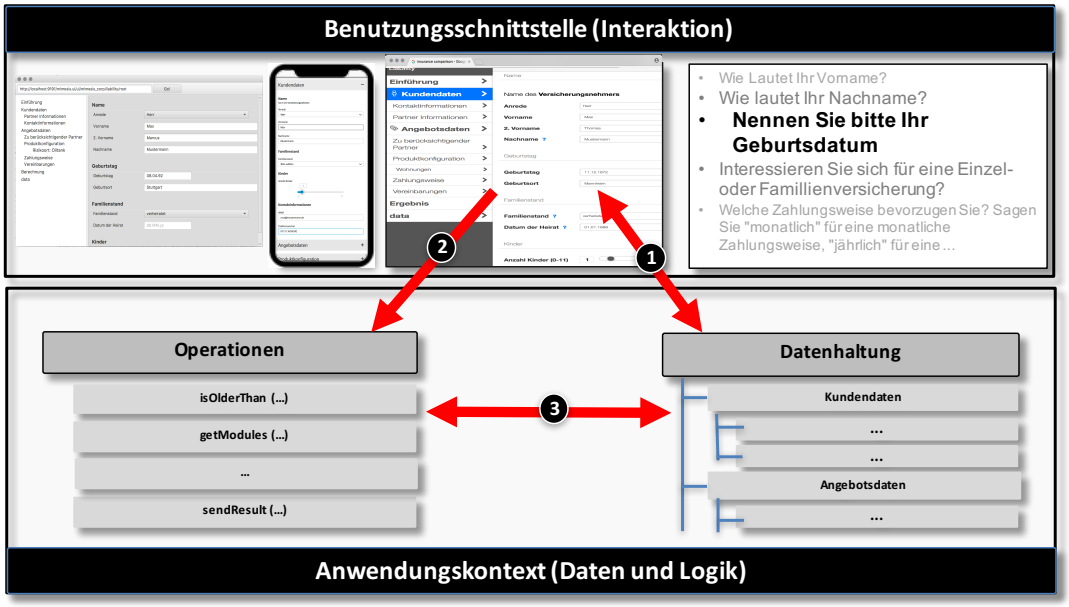


Abbildung 5.2. Komponenten und Architektur dialogbasierter Anwendungen

Die **Benutzungsschnittstelle** stellt die Interaktionsmöglichkeiten des Nutzers mit der Anwendung zur Verfügung. In Abbildung 5.2 ist dies exemplarisch für grafische und sprachbasierte Eingaben dargestellt.

Die von der Benutzungsschnittstelle zur Laufzeit benötigten Daten und Operationen können als **Anwendungskontext** aufgefasst werden, mit dem die Benutzungsschnittstelle kommuni-

ziert. Die darin enthaltene **Datenhaltung** verwaltet die Eingaben des Nutzers und repräsentiert den Zustand der Anwendung. Im Verlauf der Nutzung der Anwendung liest die Benutzungsschnittstelle Daten aus der Datenhaltung und stellt diese für den Nutzer dar. Eingegebene Daten werden in die Datenhaltung geschrieben ① und verändern dadurch den Zustand. Im Verlauf der Nutzung entsteht dadurch eine Datenstruktur, die sukzessive durch den Nutzer erweitert wird. Während der Nutzung wird Programmlogik in Form von **Operationen** angestoßen ②, welche Einfluss auf das Verhalten der Anwendung haben (z.B. zur Validierung von Eingaben oder Aufruf von Backend-Funktionalitäten). Die Operationen lesen und schreiben ebenfalls Daten der Datenhaltung ③ und verändern dadurch den Zustand der Anwendung.

Dieses Architekturmuster wird als **konzeptioneller Rahmen** für die folgenden Betrachtungen der Eigenschaften von dialogbasierten Benutzungsschnittstellen zugrunde gelegt. Im Folgenden wird daher davon ausgegangen, dass zur Laufzeit ein Anwendungskontext existiert, welcher die Datenhaltung übernimmt und Operationen der Anwendung bereitstellt.

5.3 Struktur und Aufbau der Benutzungsschnittstelle

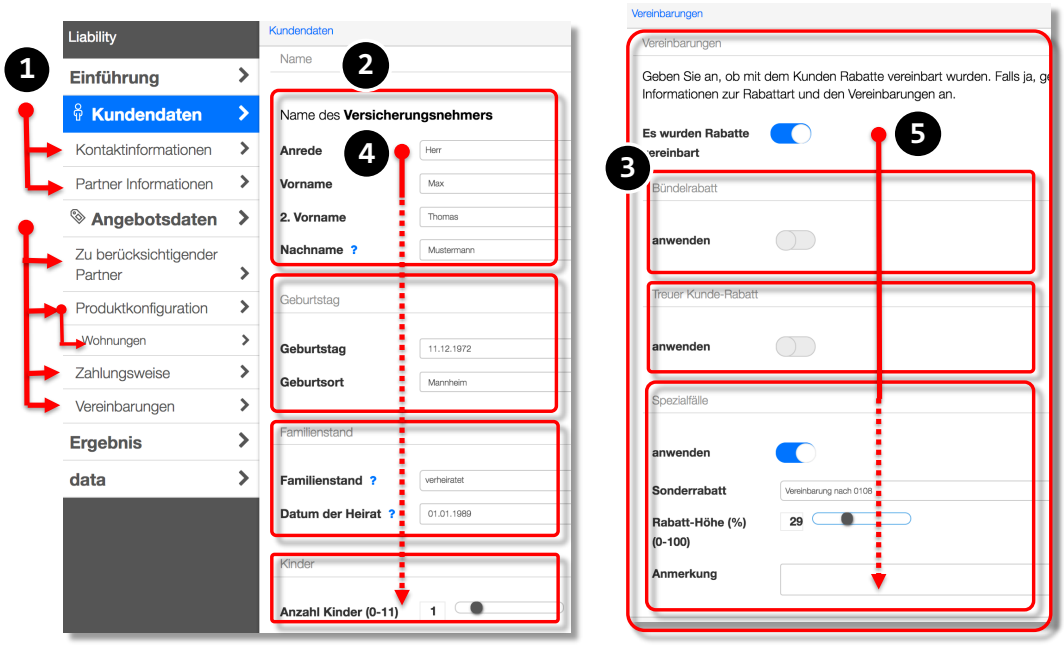
Der Aufbau von Benutzungsschnittstellen dialogbasierter Anwendungen folgt grundsätzlichen Gestaltungsprinzipien, die zu einer optimalen Nutzbarkeit beitragen [13, 89, 210]. Die Norm DIN EN ISO 9241-143 [108] gibt für Formularanwendungen Gestaltungshinweise, die durch Muster zur räumliche Anordnung von Formularelementen ergänzt werden[32]. Die Nutzbarkeit wird danach insbesondere durch die intuitiv nachvollziehbare **Gruppierung** und **Reihenfolge** von Informationen bestimmt. Fragen werden vom Anwender besser erfasst, wenn sie thematisch gruppiert sind und deren Reihenfolge bekannten Mustern folgt [41, 89]. In Benutzungsschnittstellen wird dies durch die **inhaltliche Nähe zusammengehörender Informationen** erreicht.

Abbildung 5.3 zeigt die strukturellen Eigenschaften am Beispiel der *Antragstrecke Haftpflichtversicherung* aus Abschnitt 2.2.1. Zur Strukturierung von Informationen werden Dialogseiten verwendet, die Fragen in **semantisch zusammengehörende Gruppen** gliedern. Der Navigationsbereich ① zeigt die Aufteilung z.B. in den Seiten *Einführung*, *Kundendaten* und *Angebotsdaten*. Inhalte eines Fragebereichs können dabei auf Unterseiten ausgelagert sein (z.B. *Kontaktinformationen* und *Partnerinformationen* als Teil der *Kundendaten*). Hieraus ergibt sich eine **hierarchische Struktur von Dialogseiten**, welche eine gesonderte Erfassung in sinnvollen Einheiten ermöglicht.

Die Fragen einer Seite sind in **Formularbereichen** zusammengefasst ② (z.B. *Name*, *Geburtstag*, *Familienstand* und *Kinder*), welche inhaltlich zusammengehörende Interaktionselemente gruppieren. Insbesondere bei einer großen Zahl zu erfassender, strukturierter Daten werden die Interaktionselemente in weitere, untergeordnete Formularbereiche gruppiert, die ggf. ebenfalls eine hierarchische Beziehung zueinander aufweisen ③ (z.B. Strukturierung getroffener *Vereinbarungen* in *Bündelrabatt*, *Treue Rabatt* und *Spezialfälle*).

Seiten, Formularbereiche und Interaktionselemente stehen in einer temporalen bzw. sequentiellen Beziehung zueinander und sind in einer für den Nutzer *sinnvollen* **Reihenfolge** angeordnet. In Abbildung 5.3 (a) ④ folgt z.B. die Abfrage der Namensinformationen dem bekannten Muster *Anrede, Vorname, 2.Vorname* und *Nachname*. Die Reihenfolge der *Verein-*

[32] z.B. [13] und https://entwickler.de/online/ux/ui-patterns-im-ux-design-210179.html



(a) Seitenhierarchie und Formularbereiche (b) Hierarchische Gruppierung

Abbildung 5.3. Gruppierung von Informationen

barungen ⑤ hingegen folgt keinem gängigen Muster und wird aufgrund der Fachlichkeit der Anwendung zum Entwicklungszeitpunkt bestimmt.

Zur Strukturierung der Benutzungsschnittstelle werden **Informationen zur inhaltlichen Zusammengehörigkeit der Daten** benötigt [194]. Zudem sind Informationen zur **Reihenfolge** von Fragen erforderlich. Bei der manuellen Erstellung einer Benutzungsschnittstelle werden diese Informationen implizit zur Gruppierung und Anordnung der Interaktionselemente herangezogen [41].

Die Gruppierung ergibt sich aus dem *Grad der Zusammengehörigkeit* der Daten. Die Zusammengehörigkeit ergibt sich aus den semantischen Bezügen der Fragen untereinander. In der Linguistik werden diese semantischen Bezüge unter *Kohärenz* und *Kohäsion* subsummiert und tragen maßgeblich zur Strukturierung der Benutzungsschnittstelle bei [194]. Im Folgenden wird die inhaltliche Zusammengehörigkeit als **semantische Kohäsion** bezeichnet[33]. Elemente, die eine semantische Kohäsion aufweisen, bilden eine Gruppe. Elemente, die innerhalb einer solchen Gruppe eine stärkere Bindung aufweisen, werden in Untergruppen zusammengefasst.

Hieraus ergibt sich die o.g. hierarchische Struktur von Gruppen, welche die Kohäsion der Fragen innerhalb eines Fragebereichs widerspiegeln (**innere Kohäsion**). Die innere Kohäsion bestimmt die Verteilung der Gruppen und Fragen auf Seiten. Aufgrund der Kohäsionsstär-

[33] In der Informatik werden die Begriffe Kohäsion und Kopplung ebenfalls genutzt. In einem objektorientierten System beschreibt dabei Kopplung den Grad der Abhängigkeiten zwischen den einzelnen Operationen und Objekttypen, Kohäsion den Grad des Zusammenhalts innerhalb einer Operation bzw. eines Objekttyps [201]. Aufgrund der Anwendung der Begriffe im UI-Umfeld wird im Folgenden jedoch die linguistische Sicht aus [194] verwendet.

ke kann entschieden werden, an welchen Stellen der Fragefluss unterbrochen und auf einer neuen Seite fortgesetzt werden kann. Dadurch wird vermieden, dass zusammengehörende Gruppen aufgebrochen werden (z.B. die Abfrage von *Straße/Hausnummer* auf der einen und *Postleitzahl/Ort* auf einer Folgeseite).

Ob eine Gruppe von Fragen auf eine Unterseite ausgelagert werden kann, hängt hingegen von der *Kohäsion einer Gruppe zu ihrem Umfeld* ab (**äußere Kohäsion**). Sie beschreibt die Abhängigkeit der Fragen einer Untergruppe zu denen der übergeordneten Gruppe. Untergruppen mit einer **starken äußeren Kohäsion** können dabei nicht aus dem Fragefluss entfernt werden, da sie semantisch vom Umfeld abhängen. Gruppen mit einer **schwachen äußeren Kohäsion** können hingegen herausgelöst werden.

Unterschiede in Anwendungsvarianten

Inhaltliche Varianten unterscheiden sich im *Umfang der Fragen* (vgl. Abschnitt 2.3.2). Zur Beschreibung einer inhaltlichen Variante sind daher Informationen darüber erforderlich, welche Fragegruppen und Fragen einer Anwendung für eine Variante irrelevant sind.

Technische Varianten unterscheiden sich in einer dem Kontext angepassten Seitenaufteilung und Gruppierung der Daten. Insbesondere folgende Rahmenbedingungen bestimmen die Präsentation:

- Menge darstellbarer Informationen auf einer Seite
- Verteilung der Fragen / Gruppen auf Seiten und Unterseiten [151]
- Verfügbare Navigationsformen [217]

Abbildung 5.4 zeigt diese Unterschiede am Beispiel aus Abschnitt 2.2.1 zur Erfassung der *Kontaktinformationen* der *Personendaten*.

Auf Geräten mit geringen Größenrestriktionen können größere Datenmengen pro Seite hierarchisch strukturiert erfasst werden. Daten werden ggf. auf Unterseiten verteilt, die über eine hierarchische Navigation frei gewählt werden können. In Abbildung 5.4 ist dies anhand der Desktop-Variante für einen Vertreter dargestellt. Die Seiteninhalte werden als deutlich abgegrenzte Gruppen präsentiert, welche die hierarchischen Beziehungen hervorhebt. Die *Kontaktinformationen* sind als eigenständige Unterseite der *Personendaten* herausgelöst und in der Navigation anwählbar.

Geräte mit starken Größenrestriktionen (z.B. Mobiltelefone, Tabletcomputer) hingegen stellen weniger Interaktionselemente auf einer Seite dar. Die Seiten werden meist sequentiell durchlaufen (sequentielle Navigation). Gruppen auf einer Seite weisen eine geringe hierarchische Struktur auf. Abbildung 5.4 zeigt dies anhand der mobilen Variante, die die Seiten der Anwendung als faltbare Bereiche präsentiert. Die Navigation erfolgt durch sequentielle Anwahl der Bereiche. Die Inhalte der Seite sind im Vergleich zur Desktop-Variante weniger umfangreich und flacher strukturiert. Die in der Desktop-Variante als Unterseite erfassten *Kontaktinformationen* werden hier im Fragefluss der *Personendaten* direkt nach den Informationen zu Kindern dargestellt. Sie erscheinen auf der selben hierarchischen Ebene wie die restlichen Personendaten.

Technische Varianten unterscheiden sich in einer dem Interaktions- und Technologiekontext angepassten Seitenaufteilung und angepassten Darstellung der Hierarchietiefe für Gruppen.

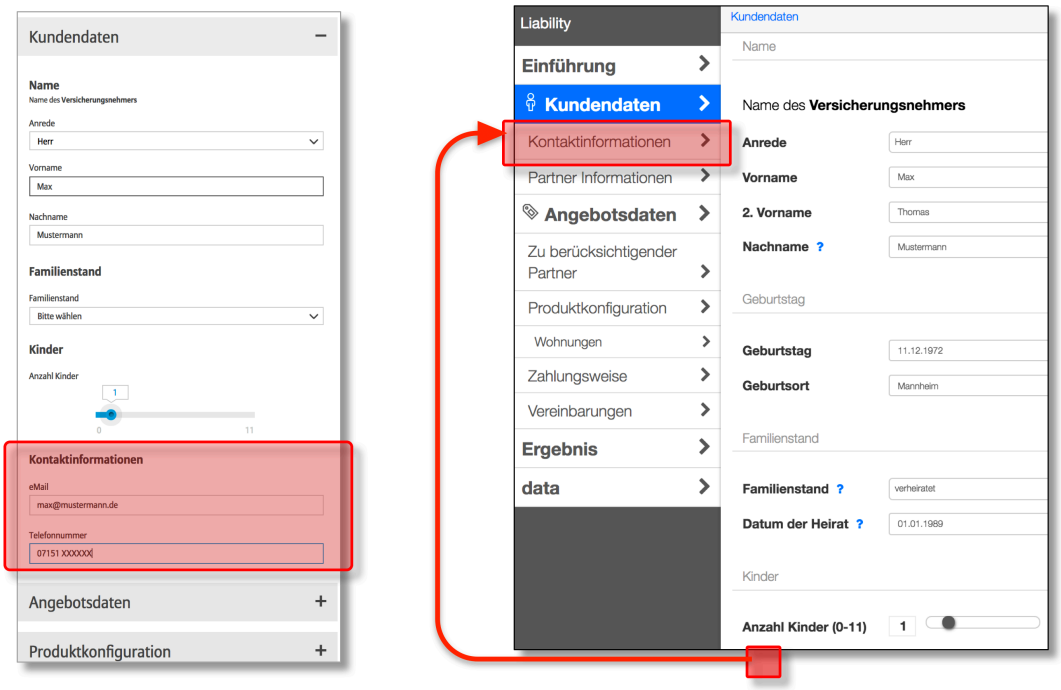


Abbildung 5.4. Strukturunterschiede von Varianten

Diese Anpassungen ergeben sich aus der **semantischen Kohäsion** der Daten, die abhängig vom Interaktions- und Technologiekontext lediglich unterschiedlich genutzt wird.

Die Seitenaufteilung für eine technische Variante erfolgt aufgrund der inneren Kohäsion. Plattformspezifische Regeln bestimmen dabei den möglichen Umfang darstellbarer Interaktionelemente, die innere Kohäsion die möglichen Trennpunkte. Weitere Regeln bestimmen die mögliche hierarchische Darstellungstiefe und Verteilung auf Unterseiten, die innere und äußere Kohäsion die Gruppenstruktur und potentiell herauslösbare Untergruppen. Darüber hinaus sind keine weiteren Informationen erforderlich.

Anforderungen an die Modellierung

Tabelle 5.1 fasst die beobachteten Eigenschaften dialogbasierter Benutzungsschnittstellen hinsichtlich Struktur und Aufbau zusammen. Unter (IR.1)-(IR.4) werden die strukturellen Eigenschaften aufgeführt. Die rechte Spalte fasst die Informationen zusammen, die zur Beschreibung der Eigenschaften identifiziert wurden.

Zur Präsentation werden inhaltlich zusammengehörende Fragen in Darstellungseinheiten und Gruppen zusammengefasst, die hierarchisch strukturiert sind und die räumliche Nähe semantisch verwandter Informationen beschreiben **(IR.1)**. Im Modell werden Informationen zum Grad der Zusammengehörigkeit der Daten (**semantische Kohäsion**) benötigt, die einerseits den Grad der Kohäsion innerhalb einer Gruppe (**innere Kohäsion**) andererseits der Gruppe zu ihrem Umfeld (**äußere Kohäsion**) beschreibt.

Innerhalb einer Gruppe weisen enthaltene Untergruppen und Fragen eine für den Nutzer sinnvolle Reihenfolge auf **(IR.2)**. Zur automatischen Herleitung werden im Modell Informationen zur **temporalen Abfolge** benötigt, in der die Fragen präsentiert werden.

ID	Eigenschaften der Benutzungsschnittstelle	Anforderungen an die Modellierung
Modellierung Struktur / Aufbau		
(IR.1)	■ Gruppierung von Fragen in Dialogbereichen ■ Hierarchische Inklusion von Fragegruppen in Dialogbereichen ■ Räumliche Nähe zusammengehörender Informationen	■ **Gruppierung** semantisch zusammenhängender Elemente ■ **Innere Kohäsion**: Grad der semantischen Bindung der enthaltenen Elemente einer Gruppe ■ **Äußere Kohäsion**: Grad der semantischen Bindung einer Gruppe zum Umfeld
(IR.2)	Sinnvolle Reihenfolge von ■ Fragen und Gruppen im Fragefluss ■ Seiten/Unterseiten der Navigation	■ **Temporale Abfolge** der Gruppen- und Datenelemente
Technische Varianten		
(IR.3)	■ Semantisch sinnvolle Verteilung von Fragen auf Seiten ■ Steuerung der Tiefe der hierarchischen Darstellung (Unterseiten bzw. eingebettet) ■ Herleitung verschiedener Navigationsarten (z.B. sequentiell, hierarchisch)	■ Benötigte Informationen zur Modellierung identisch mit (IR.1)
Inhaltliche Varianten		
(IR.4)	Variantenspezifische Auswahl relevanter Fragen abhängig von ■ Nutzungskontext / Nutzergruppe ■ Technologiekontext / Plattform	■ **Identifizierbarkeit** von Elementen ■ **Auswahl relevanter Elemente** für spezifische Varianten

Tabelle 5.1. Anforderungen und erforderliche Informationen (Struktur/Aufbau)

Technische Varianten unterscheiden sich in ihrer Seitenaufteilung, Hierarchietiefe der dargestellten Gruppen und Navigationsformen **(IR.3)** . Die zur Herleitung erforderlichen Informationen entsprechen den unter (IR.1) genannten zur **semantischen Kohäsion**.

Inhaltliche Varianten unterscheiden sich in der Auswahl für eine Variante relevanter Fragen **(IR.4)**. Abhängig vom Interaktions- und Technologiekontext müssen **relevante Fragen bzw. Gruppen im Modell identifiziert** werden können, die in einer Variante zu berücksichtigen sind. Das Modell muss eine solche Identifizierung und Auswahl ermöglichen.

5.4 Typisierte Ein-/Ausgabe von Daten

Neben dem Aufbau der Benutzungsschnittstelle beeinflusst die Auswahl der Interaktionselemente die Bedienbarkeit einer Anwendung. Hier steht die intuitive Nachvollziehbarkeit der Interaktionselemente im Vordergrund [107, 210, 222]. Tidwell [222] nennt folgende Kriterien, welche die Auswahl beeinflussen:

- Erfahrung der Nutzergruppe im Umgang mit entsprechenden Benutzungsschnittstellen
- Erwartungen aufgrund ähnlicher Anwendungen
- Domänenwissen der Nutzergruppe
- technologische und räumliche Restriktionen

Ein Interaktionselement soll dem Nutzer nachvollziehbar anzeigen, welche Art von Information erwartet wird und ihn zielgerichtet bei der Eingabe unterstützen [107]. Dies kann durch die Verwendung von Elementen erreicht werden, die der Nutzer bereits kennt und erwartet (*Erfahrung*, *Erwartungen*) oder die aufgrund des Anwendungsfalls intuitiv erfassbar sind (*Domänenwissen*). In den betrachteten Anwendungen konnten zwei Arten von Interaktionselementen identifiziert werden:

- Standard-Interaktionselemente
- Applikationsspezifische Interaktionselemente

Standard-Interaktionselemente für grundlegende Datentypen

Standard-Interaktionselemente treten am häufigsten auf und erfassen Daten mit einem grundlegenden Datentyp (z.B. *Text, natürliche/reelle Zahlen, Datum, Zeit, Boolsche Werte, 1-aus-N-Auswahl, M-aus-N-Auswahl*). Für diese Typen existieren in allen Technologien standardisierte Interaktionselemente (*Controls, Widgets*), welche domänenunabhängig sind und die Interaktion für die Plattform optimiert umsetzen. In Tidwell [222, S. 210 ff] findet sich eine Zusammenstellung in Formularanwendungen üblicherweise genutzter Typen. Diese Typen können über zusätzliche Eigenschaften (*Restriktionen*) näher beschrieben sein. Übliche Restriktionen in Formularanwendungen sind z.B.[34]:

- Syntaktische Muster bei der Eingabe von textuellen Angaben
- Beschränkung möglicher Werte zur Auswahl
- Minimal-/Maximalwerte für numerische Eingaben
- Schrittweiten bei numerischen Angaben
- Minimale/maximale Anzahl möglicher Zeichen bei der Eingabe
- Beschränkung der Nachkommastellen bei reellen Zahlen

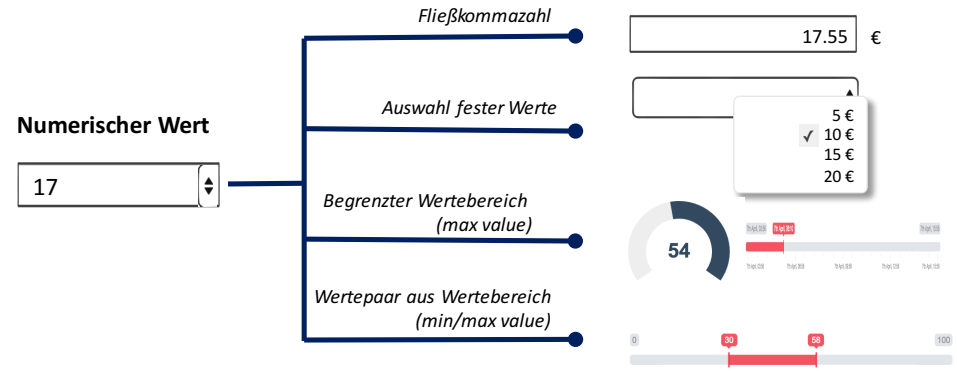
Die Auswahl eines Interaktionselements erfolgt abhängig vom **Typ und den Restriktionen** des zu erfassenden Datums. In Abbildung 5.5 ist die Auswahl am Beispiel von numerischen Werten (*number*), Wertebereichen (*numberrange*) und einer *M-aus-N-Auswahl* exemplarisch dargestellt.

Für einen ganzzahligen, numerischen Wert ohne Restriktion wird ein einfaches Eingabefeld verwendet, welches sowohl die manuelle Eingabe als auch das Erhöhen/Vermindern des Wertes mittels Schaltern ermöglicht (s. Abbildung 5.5 (a)). Wird der Typ über die Angabe einer *festen Zahl an Dezimalstellen* eingeschränkt, wird eine formatbeschränkte Eingabe verwendet. Existiert eine *Minimal-/Maximalwert*-Restriktion mit einer festen *Schrittweite*, wird ein Schiebe- bzw. Drehregler verwendet. Abbildung 5.5 (b) zeigt die Wahl eines Interaktionselements für eine *M-aus-N-Auswahl*, die auf eine Menge möglicher Werte beschränkt ist. Ist die Zahl möglicher Werte gering, erfolgt die Eingabe über Optionsfelder. Ist deren Zahl hoch, wird ein Suchfeld zur Auswahl von Werten im Wertebereich angeboten.

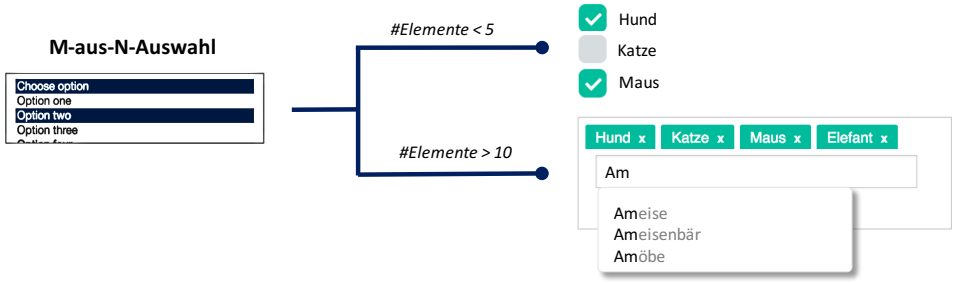
Die gewählten Interaktionselemente schränken die Auswahl möglicher Werte von vornherein ein, wodurch Fehleingaben vermieden werden. Die Auswahl eines geeigneten Interaktionselements durch den Entwickler erfolgt nicht willkürlich. Üblicherweise existieren hierfür Plattform-Standards und Unternehmens-Styleguides mit Regeln für deren Verwendung. Zur Auswahl eines geeigneten Standard-Interaktionselements werden lediglich folgende Informationen benötigt:

- **Typ des zu erfassenden Datums**
- **geltende Restriktionen**

[34] XML-Schema [61] definiert beispielsweise 14 Restriktionsarten (*restricting facets*)



(a) Zahleneingabe nach Typ und Restriktionen



(b) M-aus-N-Auswahl nach Anzahl enthaltener Elemente

Abbildung 5.5. Auswahl konkreter Ausprägungen

Bei der Erstellung der Benutzungsschnittstelle bestimmen dann **plattformspezifischen Regeln** die Auswahl des konkreten Interaktionselements.

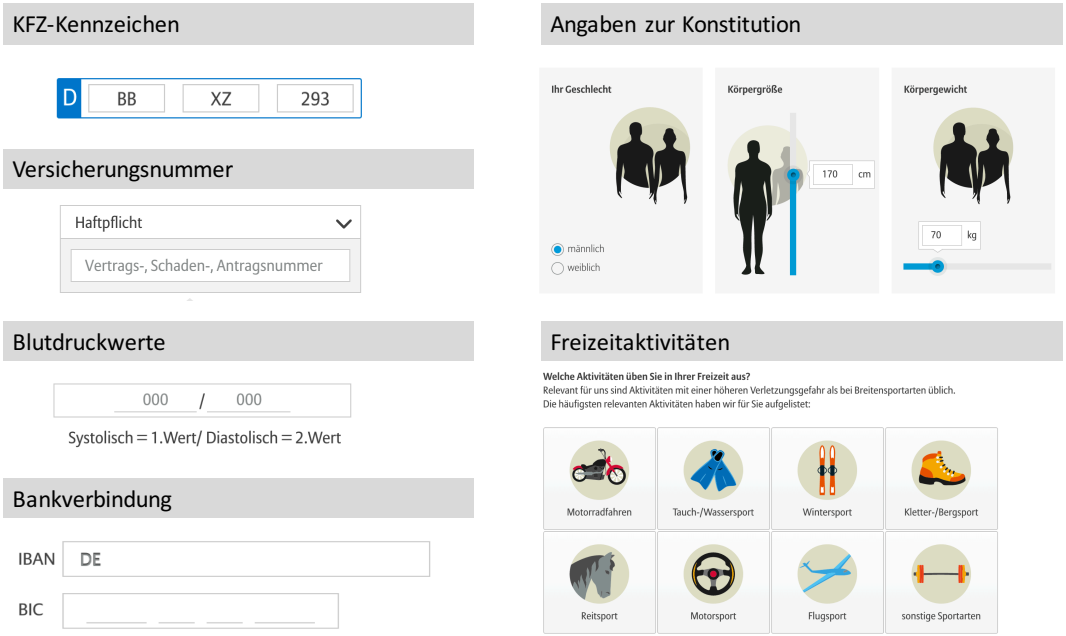
Applikationsspezifische Interaktionselemente

Zur Eingabe von Werten werden häufig spezialisierte Interaktionselemente verwendet, die von der fachlichen Semantik des Datums abhängen und die Eingabe über Standards hinaus unterstützen. Beispiele hierfür reichen von der Eingabe einer E-Mail-Adresse, einer IBAN/BIC-Kombination, einer strukturierten Vertragsnummer bis hin zur Auswahl von Körperregionen zur Erfassung von Gesundheitsfragen. Diese sind für einen spezifischen Interaktions- und Technologiekontext optimiert und ermöglichen eine intuitive Erfassung des Datums, z.B. durch

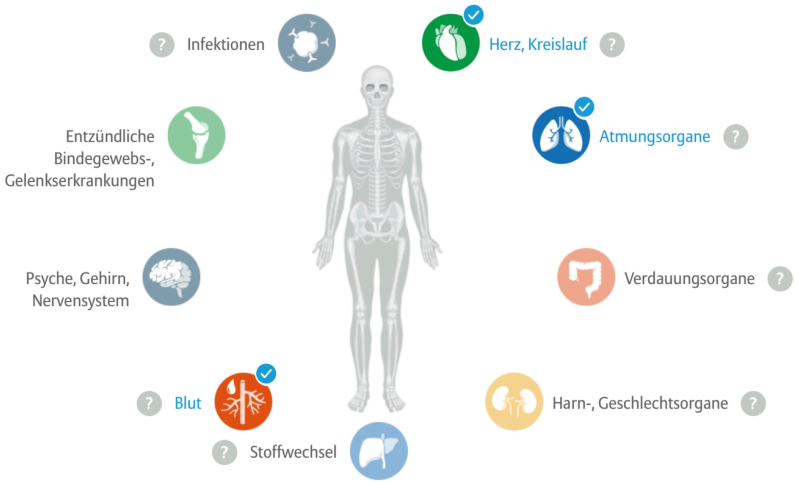
- Strukturierte Interaktionselemente zur Eingabe eines Wertes
- Verwendung bekannter Metaphern zur Darstellung
- Spezialisierte Interaktionselemente
- Zusammenfassung/Kombination von Werten in einem Interaktionselement

Strukturierte Interaktionselemente beschränken die Eingabemöglichkeiten der Daten und reduzieren damit die Wahrscheinlichkeit von Fehleingaben. Abbildung 5.6 (a) zeigt Beispiele für KFZ-Kennzeichen, Versicherungsnummern, Blutdruckwerte und Bankverbindungen über IBAN/BIC. Die möglichen Nutzereingaben werden formatiert dargestellt und ggf. auf sinnvolle Werte beschränkt. Die *Verwendung bekannter Metaphern* (*interface meta-*

phors[35] [8]) zur Darstellung ermöglicht hingegen die schnelle Erfassung der Bedeutung einer Frage ([56, 217]). Als Beispiel ist in Abbildung 5.6 (a) die Erfassung von Daten zur Konstitution einer Person (*Geschlecht, Größe, Gewicht*) bzw. ausgeübter Freizeitaktivitäten dargestellt. Hier werden Abbildungen verwendet, die dem Nutzer bekannt sind und so eine schnelle Assoziation des fachlichen Inhalts ermöglichen.



(a) Strukturierte Eingabe und Metaphern



(b) Komplexes Interaktionselement: Körperatlas

Abbildung 5.6. Applikationsspezifische Interaktionselemente

Spezialisierte Interaktionselemente nutzen Domänen- und Erfahrungswissen des Nutzers zur Darstellung. Abbildung 5.6 (b) zeigt ein Interaktionselement zur Angabe erkrankter Körperbereiche bei der Erfassung von Gesundheitsfragen. Dem Nutzer werden Körperbereiche als

[35] https://en.wikipedia.org/wiki/Interface_metaphor

Körperatlas angezeigt, die durch die räumliche Anordnung eine intuitive Auswahl ermöglichen. Die Spezialisierung kann soweit gehen, dass für bestimmte Angaben *kombinierte Interaktionselemente* verwendet werden. Hierzu werden zusammengehörende Werte an der Oberfläche in einem Interaktionselement zusammengeführt. In Abbildung 5.6 (a) ist das am Beispiel der Angabe der Konstitution dargestellt, in welcher Geschlecht, Größe und Gewicht in einem Interaktionselement zusammengeführt werden.

Die Auswahl applikationsspezifischer Interaktionselemente erfolgt aufgrund folgender Informationen:

- **Typ des zu erfassenden Datums**
- **geltende Restriktionen**
- **fachliche Semantik des erfragten Datums** bzw.
- **fachliche Semantik einer Gruppe von Daten**

Bietet der Technologiekontext für die spezifische Semantik oder die Kombination von Datenfeldern ein spezielles Interaktionselement, wird dieses gewählt.

Unterschiede in Anwendungsvarianten

Inhaltliche Varianten von Interaktionselementen existieren dann, wenn der *Nutzungskontext* der Anwendung einen Einfluss auf den Typ des zu erfassenden Datums oder anzuwendende Restriktionen besitzt. Ein Beispiel ist die länderspezifische Eingabe einer Postleitzahl, die in Deutschland als fünfstelliges Zahlenfeld, in Großbritannien als formatbeschränktes Textfeld erfolgt.

Für die Erstellung von inhaltlichen Varianten sind Informationen darüber erforderlich, wie sich Eigenschaften der Daten verändern. Es müssen **Unterschiede am Typ, an den Restriktionen und der Semantik eines Elements** angegeben werden können, die abhängig vom Nutzungskontext variieren.

Technische Varianten von Interaktionselementen hängen in hohem Maße vom Interaktions- und Technologiekontext der Anwendung ab. Jede Zielplattform verwendet eigene Darstellungen für die zu erfassenden Daten, die sich in der Bedienung z.T. beträchtlich unterscheiden. Beispielsweise verwenden mobile Endgeräte Interaktionselemente, die auf die Bedienung mit Touchscreens ausgelegt sind (vgl. [51, 199]). Auf Desktop-Geräten hingegen erfolgt die Eingabe mit Tastatur und Maus. Die räumlichen Restriktionen bestimmen zudem, wie komplex die Interaktionselemente gestaltet werden können.

Bei **Standard-Interaktionselementen** ist die Varianz gering. Hier existieren für jeden Interaktions- und Technologiekontext standardisierte Interaktionselemente, die abhängig vom Typ und geltender Restriktionen zu verwenden sind. Die Auswahl erfolgt lediglich aufgrund **vom Interaktions- und Technologiekontext vorgegebener Regeln**.

Bei **applikationsspezifischen Interaktionselementen** bestehen größere Unterschiede in der Darstellung. Da die Eingabeform fachlich bestimmt ist und ggf. komplexe Zusammenhänge abgebildet werden, existieren für unterschiedliche Interaktions- und Technologiekontexte individuelle Varianten. Abbildung 5.7 (links) zeigt dies am Beispiel des spezialisierten Interaktionselements *Körperatlas*. Die Desktop-Variante wird in der mobilen Variante vereinfacht und als Liste von Schaltflächen dargestellt. Abbildung 5.7 (rechts) zeigt die Varianz

für ein zusammengesetztes Interaktionselement zur Angabe der *Konstitution*. In der mobilen Variante wird das zusammengesetzte Interaktionselement in Einzelfragen zerlegt und die Zusammenfassung aufgelöst.

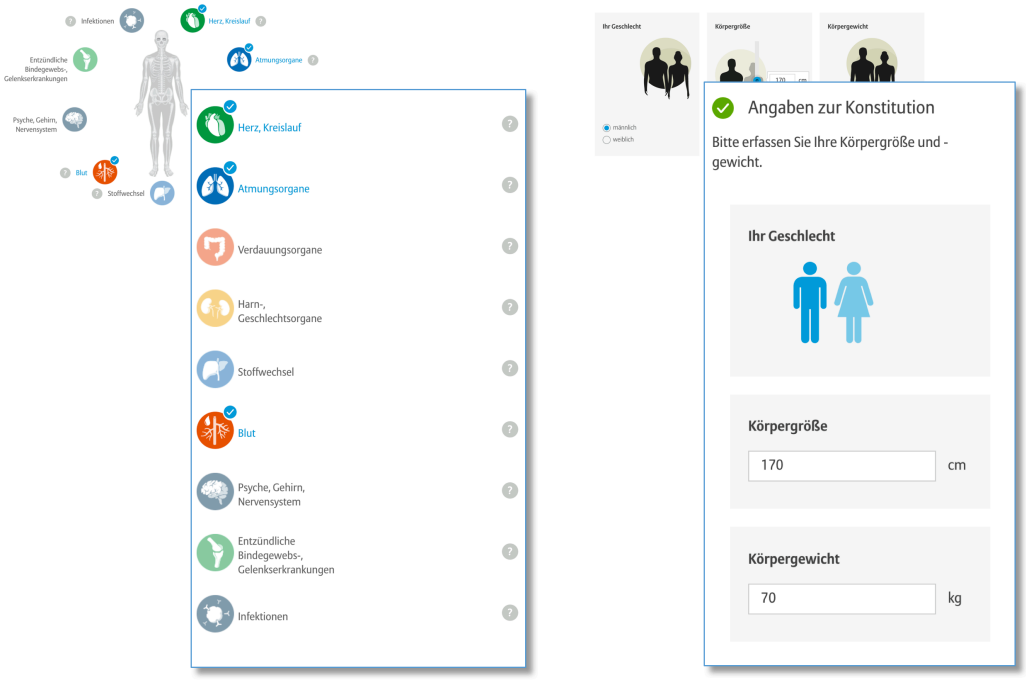


Abbildung 5.7. Plattformspezifische Varianten

Beim Vergleich technischer Varianten applikationsspezifischer Interaktionselemente während der Analyse war eine Auffälligkeit zu beobachten. Es wurden nicht durchgängig in allen Varianten spezialisierte Elemente verwendet. In einigen Varianten wurde **auf Standard-Interaktionselemente zur Darstellung zurückgegriffen, obwohl in anderen Varianten der Anwendung spezialisierte Elemente verwendet** wurden. Hieraus konnte eine besondere Eigenschaft abgeleitet werden: so komplex die Interaktionselemente auch scheinen, sie erfassen Daten, die analog zu den Standard-Interaktionselementen auf einen grundlegenden Datentyp zurückzuführen sind. Abbildung 5.8 illustriert dies exemplarisch am *Körperatlas* aus Abbildung 5.6 (b).

Aufgrund der semantischen Information, dass es sich um Körperregionen handelt, kann in der Desktop-Variante der Anwendung ein komplexes Interaktionselement verwendet werden. Wird die fachliche Semantik ignoriert, kann als Standard-Interaktionelement eine einfache Auswahlliste verwendet werden.

Trotz der erheblichen Unterschiede in der Darstellung, werden zur Auswahl einer technischen Variante eines applikationsspezifischen Interaktionselements **keine zusätzlichen Informationen** benötigt.

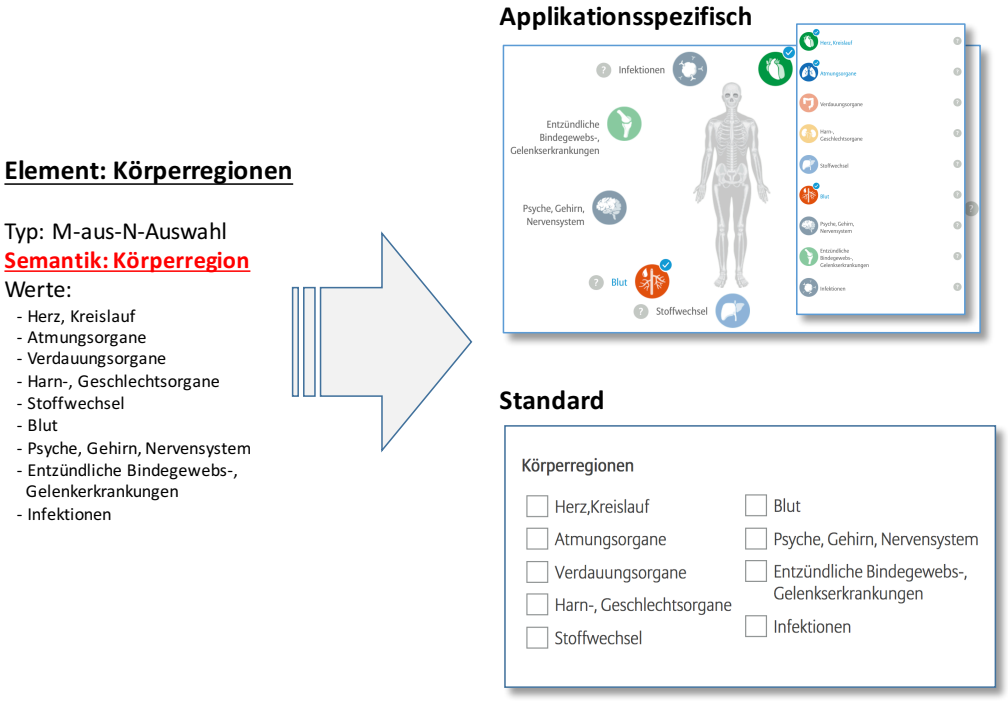


Abbildung 5.8. Korrespondierende Standard-Interaktionselemente

Anforderungen an die Modellierung

Tabelle 5.2 fasst die beobachteten Eigenschaften zur typisierten Ein-/Ausgabe von Daten zusammen. Unter (IR.5)-(IR.9) werden die beobachteten Eigenschaften aufgeführt und die Informationen zusammengefasst, die zu deren Beschreibung erforderlich sind und damit in einem Anwendungsmodell enthalten sein müssen.

Für unterschiedliche Typen zu erfassender Daten werden standardisierte Interaktionselemente verwendet, die durch den Interaktions- und Technologiekontext bestimmt sind (IR.5). Darüber hinaus werden applikationsspezifische Interaktionselemente verwendet, sofern diese für einen Nutzungs- und Technologiekontext existieren (IR.6). Ggf. werden zudem für Gruppen applikationsspezifische Varianten verwendet, die Fragen zusammenfassen (IR.7). Zur Herleitung von Standard-Interaktionselementen sind Informationen zum **Typ und geltender Restriktionen** des zu erfassenden Datums im Modell erforderlich **(IR.5)**. Zur Herleitung applikationsspezifischer Interaktionselemente für Daten und Gruppen werden zusätzlich **Informationen zur fachlichen Semantik** benötigt **(IR.6)** und **(IR.7)**.

Die Herleitung technischer Varianten erfordert die Festlegung von **Auswahlregeln zur Bestimmung geeigneter Interaktionselemente** abhängig vom Interaktions- und Technologiekontext **(IR.8)**. Generatoren entscheiden mittels dieser Regeln und der in (IR.5), (IR.6) und (IR.7) dargestellten Informationen, welches Interaktionselement erzeugt wird. Zur Bildung inhaltlicher Varianten müssen abhängig vom Nutzungskontext **alternative Eigenschaften eines Elements (Typ, Restirktionen)** modelliert werden können **(IR.9)**.

ID	Eigenschaften der Benutzungsschnittstelle	Anforderungen an die Modellierung
Modellierung typisierte Ein-/Ausgabe		
(IR.5)	Verwendung intuitiv erfassbarer Interaktionselemente zur typisierten Eingabe von Daten	■ **Typisierung** der zu erfassenden Daten ■ Spezifikation geltender **Restriktionen**
(IR.6)	Verwendung applikationsspezifischer Interaktionselemente	■ **Fachliche Semantik von Daten**
(IR.7)	■ intuitiv erfassbare Darstellung von Gruppen ■ Zusammenfassung von Daten einer Gruppe zu komplexen Interaktionselementen	■ **Fachliche Semantik von Gruppen**
Technische Varianten		
(IR.8)	Verwendung plattformspezifischer Interaktionselemente basierend auf ■ Plattform- und ■ Usability-Styleguides	■ identisch mit (IR.5)-(IR.7) ■ **Plattformspezifische Regeln zur Auswahl von Interaktionselementen** ■ **Rückfall-Option auf einfache Interaktionselemente**
Inhaltliche Varianten		
(IR.9)	Kontextabhängige Typen und Restriktionen für Interaktionselemente	■ **Alternativer Typ des zu erfassenden Datums** ■ **Alternative Restriktionen**

Tabelle 5.2. Anforderungen und Informationsbedarf (typisierte Ein-/Ausgabe)

5.5 Verhalten der Benutzungsschnittstelle

Die DIN EN ISO 9241-110 [107] führt als Grundsätze für eine ergonomische Dialoggestaltung die *Aufgabenangemessenheit*, *Steuerbarkeit*, *Erwartungskonformität* und *Fehlertoleranz* eines Systems an. Das Verhalten der Anwendung soll den Nutzer intuitiv, zielgerichtet durch die Anwendung führen und bei der Eingabe korrekter Werte unterstützen. Dies kann dadurch erreicht werden, dass z.B. im Fragefluss abhängig von bisherigen Eingaben nur relevante Fragen gestellt werden, Fehleingaben frühzeitig angezeigt (DIN EN ISO 9241-13 [104]) und lediglich valide Werte zur Auswahl angeboten werden (DIN EN ISO 9241-11 [110]). In den untersuchten Anwendungen wurden folgende Arten von Verhalten der Benutzungsschnittstelle identifiziert:

- Steuerung der Sicht-/Editierbarkeit von Informationen
- Validierung eingegebener Daten
- Elementreaktionen bei Änderung des Anwendungskontexts
- Elementinitiierte Aktionen
- Nutzer- und Systeminitiierte Aktionen

Die Analyse der o.g. Verhaltensarten ergab, dass sie durch Informationen zu **Auslöser**, **Eingangsparametern**, **Operation** und **Wirkung** beschreibbar sind. Jedes Verhalten hat einen **Auslöser**, der eine Aktion zur Folge hat. Dies kann z.B. die Änderung eines Wertes im Anwendungskontext oder das explizite Betätigen einer Schalters sein. Hierbei wird eine **Operation** ausgelöst, die als **Eingangsparameter** Daten aus dem Anwendungskontext nutzt und ggf. modifiziert (**Wirkung**).

Steuerung der Sicht-/Editierbarkeit von Informationen

In DIN ISO ISO9241-110 [107] wird als Maßnahme zur Erreichung der *Aufgabenangemessenheit* einer Anwendung angeführt, dass lediglich Informationen angezeigt werden, die im

aktuellen Zustand relevant sind. Aufgrund bereits erfasster Daten werden hierzu Informationen ausgeblendet bzw. deaktiviert.

Abbildung 5.9 zeigt dies am Beispiel der Eingabe des Familienstandes bei der Erfassung persönlicher Daten. Wird als *Familienstand* der Wert *verheiratet* gewählt, werden *Datum der Heirat* sowie *Partnerdaten* zur Eingabe angeboten. In der Navigation der Anwendung erscheint zudem der Eintrag *Partnerdaten*. Wählt der Nutzer hingegen *ledig* aus, werden die Fragen zum *Datum der Heirat* und *Partnerdaten* aus dem Fragefluss und der Navigation entfernt (Abbildung 5.9, rechts).

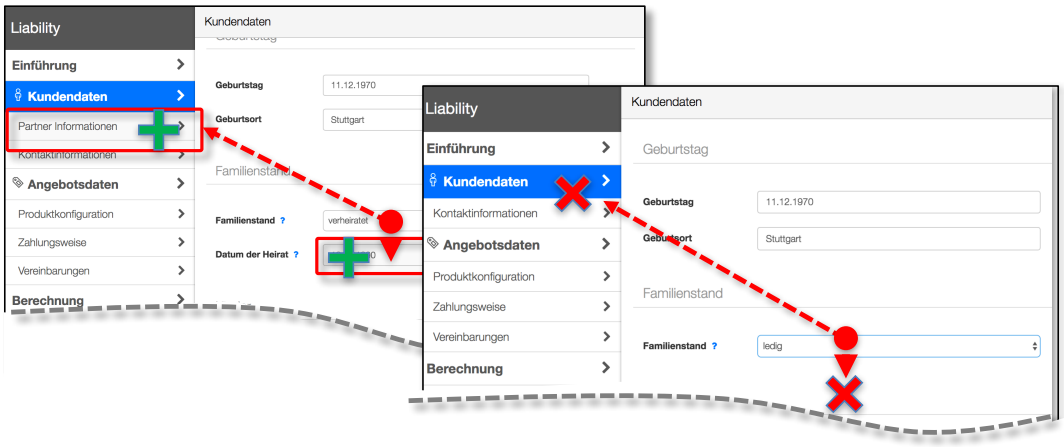


Abbildung 5.9. Änderung der Sichtbarkeit von Elementen

Das Beispiel zeigt den Bezug zum aktuellen Zustand der Anwendung. Die Sichtbarkeit der Fragen ergibt sich aus dem Inhalt des Datums *Familienstand* im Anwendungskontext. Diese Information wird erst zur Laufzeit angegeben und führt zur beschriebenen Anpassung der Benutzungsschnittstelle.

Auslöser für die Änderung der Sicht-/ Editierbarkeit eines Elements sind Änderungen im Anwendungskontext. Aufgrund der Änderung wird Programmlogik ausgeführt, die als Wirkung die Sicht- bzw. Editierbarkeit des Elements selbst verändert.

Validierung eingegebener Daten

Die Validierung eingegebener Daten wird eingesetzt, um frühzeitig Fehleingaben bzw. Inkonsistenzen der eingegebenen Daten zu vermeiden. Die Norm DIN ISO 9241-13 [104] definiert Fehler als *"Nicht-Übereinstimmung zwischen dem Ziel des Nutzers und der Reaktion des Systems"*. Um nachgelagerte Fehler zu vermeiden, muss die Benutzungsschnittstelle durch Validierung sicherstellen, dass Aufbau und Inhalt der Eingabe korrekt und plausibel sind. Hierbei wird zwischen der **syntaktischen** und **semantischen Validierung** unterschieden [193].

Bei der **syntaktischen Validierung** wird die Eingabe für ein einzelnes Element auf strukturelle Korrektheit geprüft. Zudem wird die Einhaltung von Restriktionen überprüft (vgl. Abschnitt 5.4). Sie ist elementbezogen und benötigt keine weiteren Daten aus dem Anwendungskontext. Die **semantische Validierung** hingegen führt eine inhaltliche Prüfung der Eingabe durch. Sie validiert die Konsistenz bereits erfasster Daten und verwendet hierzu Daten aus dem Anwendungskontext [193].

Abbildung 5.10 zeigt dies am Beispiel der Angabe einer Bankverbindung über IBAN und BIC. Der Aufbau einer IBAN (International Bank Account Number) und BIC (Bank Identification Code) folgt Regeln, die validierbar sind[36]. Zur **syntaktischen Validierung** wird beim Verlassen des Feldes die Eingabe auf ein regelkonformes Muster geprüft und der Nutzer ggf. durch eine Nachricht auf Fehler hingewiesen.

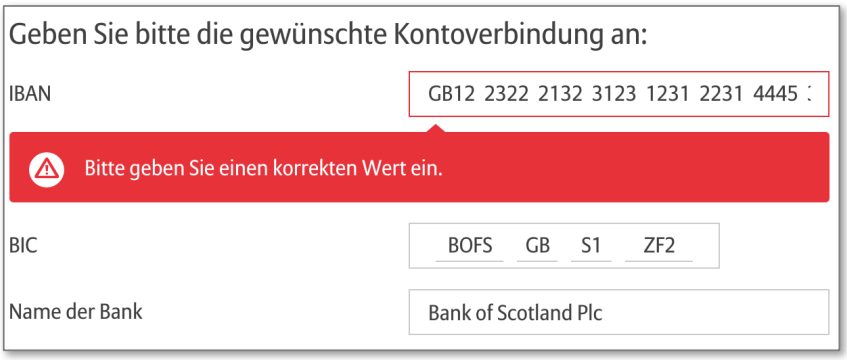


Abbildung 5.10. Fehler bei Validierung einer IBAN

Nach syntaktisch korrekter Eingabe, wird diese zusätzlich **semantisch validiert**. Dabei wird die Vollständigkeit und Plausibilität der Angaben geprüft. Hierzu kann z.B. die in der IBAN enthaltene Prüfsumme ausgewertet und deren inhaltliche Konsistenz sichergestellt werden [105].

Der Auslöser für eine Validierung ist das Element selbst. Die Eingabeparameter sind sowohl der Inhalt des Feldes (syntaktische Validierung) als auch Elemente aus dem Anwendungskontext (semantische Validierung). Zur Durchführung der Validierung werden Operationen gerufen, die als Wirkung den Validierungsstatus des Elements verändern.

Elementreaktionen bei Änderung des Anwendungskontexts

Reaktionen beschreiben das Verhalten eines Elements auf die Änderung von Inhalten anderer Elemente im Anwendungskontext. Sie werden insbesondere zur Vermeidung inkonsistenter Angaben verwendet, indem z.B. Werte sinnvoll vorbelegt oder auf mögliche Werte beschränkt werden.

Abbildung 5.11 zeigt dies am Beispiel der Vorbelegung eines Wertes. Gibt der Nutzer eine korrekte *Postleitzahl* ein, wird als Reaktion Programmlogik ausgeführt, die für den *Ort* eine passende Vorbelegung ermittelt. Die Abbildung zeigt zudem die *Beschränkung eines Elements* auf sinnvolle Werte. Das Interaktionselement für die Angabe der *Straße* reagiert ebenfalls auf die Eingabe einer *Postleitzahl*, und bietet lediglich unter der Postleitzahl verzeichnete Straßennamen an.

Auslöser für Reaktionen ist die Änderung eines Datums im Anwendungskontext zur Laufzeit, das einen Einfluss auf das reagierende Element hat. Das Element reagiert bei Änderung mit der Ausführung von Programmlogik, die als Eingabe Daten aus dem Anwendungskontext nutzt und auf den Inhalt bzw. die Restriktionen des reagierenden Elements wirkt.

[36] vgl. ISO 13616-1 [105] bzw. ISO 9362 [109]. https://www.iban.de/

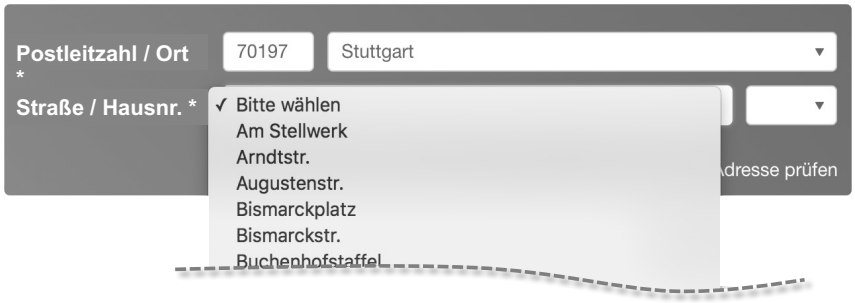


Abbildung 5.11. Anpassung von Inhalten als Reaktion auf Änderungen

Elementinitiierte Aktionen

Elementinitiierte Aktionen sind Operationen, die bei Änderung eines Elements ausgeführt werden und Auswirkung auf den Anwendungskontext haben. Sie werden angewendet, wenn die Änderung eines Feldes Aktionen erfordert, die auf den Anwendungskontext oder entfernte Systeme wirken.

Abbildung 5.12 zeigt dies am Beispiel eines Feldes zur Berufsauswahl. Bereits während der Eingabe werden Vorschläge zur Auswahl angeboten. Der Nutzer gibt erste Zeichen ein, woraufhin eine Auswahlbox mit Vorschlägen erscheint, die bei der fortgesetzten Eingabe aktualisiert wird.

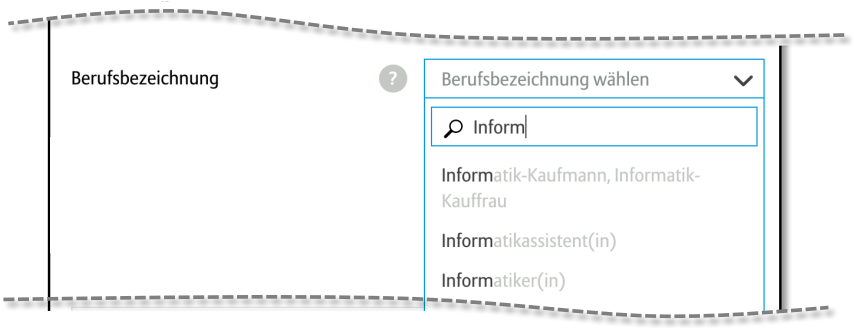


Abbildung 5.12. Vorschläge während der Eingabe als elementinitiierte Aktion

Ein Beispiel für eine komplexere elementinitiierte Aktion ist die Vorbefüllung von Vertragsdaten bei Eingabe einer validen Vertragsnummer in einem Dialog zur Änderung der versicherten Person einer Lebensversicherung. Gibt der Nutzer eine Vertragsnummer ein, wird beim Verlassen des Feldes eine Anfrage an das Bestandssystem gesendet, welche die Existenz des Vertrags prüft und Vertragsdetails zur Vorbefüllung liefert.

Auslöser einer elementinitiierten Aktion ist die Änderung des Elements. Die angestoßene Operation nutzt als Eingabeparameter Daten aus dem Anwendungskontext und wirkt auf den Anwendungskontext.

Nutzer- und systeminitiierte Aktionen

Nutzerinitiierte Aktionen sind durch den Nutzer explizit ausgelöste Operationen, die Auswirkungen auf den Anwendungskontext haben. Sie können an bestimmten Stellen im Fragefluss

optional angestoßen werden (üblicherweise über Schaltflächen oder in Menüs). *Systeminitiierte Aktionen* wirken ebenfalls auf den Anwendungskontext, werden jedoch durch externe Ereignisse ausgelöst, die in der Anwendung auftreten. Sie beschreiben Aktionen, die nicht Folge einer Änderung des Anwendungskontexts sind.

In Abbildung 5.13 (links) ist als Beispiel für eine *nutzerinitiierte Aktion* die Vorbefüllung von Daten aus dem Kundenbestand eines Vertreters dargestellt. Im Bereich *Kundendaten* wird dem Vertreter ein Menüpunkt angeboten, der die Kundensuche in seinem Bestand und die Übernahme der Daten ermöglicht. Wählt der Vertreter den Menüpunkt, wird eine Suchmaske geöffnet, welcher die Kundensuche in einem entfernten System durchführt. Wählt der Vertreter ein Ergebnis aus, werden die Daten in den Anwendungskontext übernommen.

Für eine Kundenvariante der Anwendung wird stattdessen eine *Anmelden*-Schaltfläche angeboten (Abbildung 5.13, rechts), welche eine Anmeldemaske öffnet und die Kundendaten zur Vorbefüllung lädt.

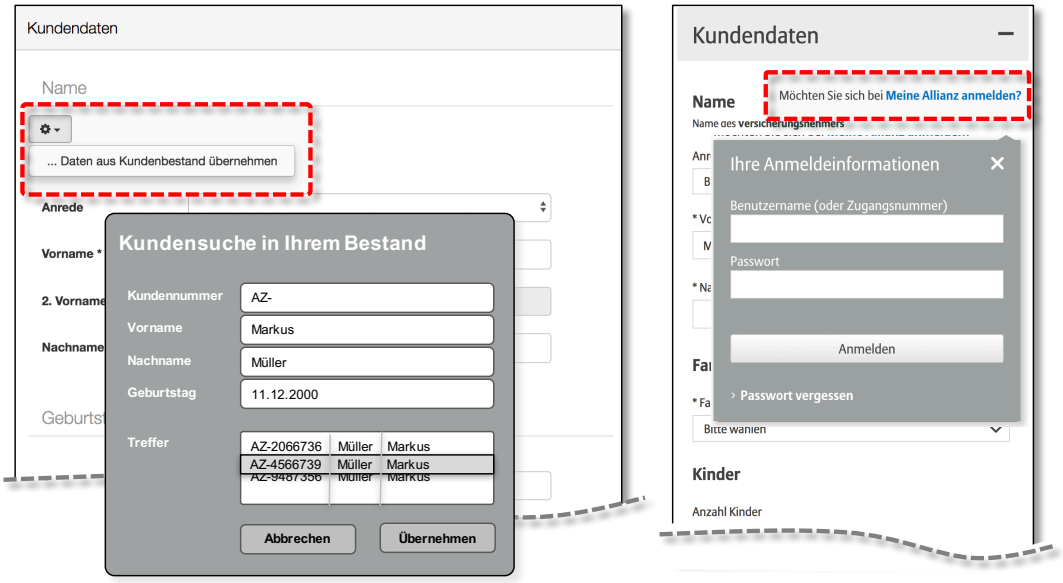


Abbildung 5.13. Nutzerinitiierte Aktionen

Die Vorbefüllung von Daten könnte als *systeminitiierte Aktion* ausgeführt werden. Eine in ein Portal eingebettet Variante der Antragstrecke wird über den Anmeldezustand des Nutzers informiert und kann auf dieses *externe Systemereignis* mit der Vorbefüllung der Daten des angemeldeten Kunden reagieren.

Die Auslöser von nutzer- und systeminitiierten Aktionen sind Ereignisse, die der Nutzer bzw. die Anwendungsumgebung auslösen. Eingabeparameter für die angestoßene Operation sind Daten aus dem Anwendungskontext. Die Operationen wirken auf den Anwendungskontext bzw. entfernte Systeme.

Unterschiede in Anwendungsvarianten

Die beschriebenen Verhaltensarten finden sich in allen betrachteten Varianten wieder. Allerdings können zwischen den Varianten Unterschiede im Verhalten auftreten, die durch den **Nutzungs- und Technologiekontext**, sowie den **variierenden Umfang der erfassten Daten** in einer Variante bestimmt sind.

Im Beispiel zur Vorbefüllung von Daten im vorangegangenen Abschnitt *Nutzer- und systeminitiierte Aktionen* wird in der Vertreter-Variante als nutzerinitiierte Aktion die Vorbefüllung der Daten aus der Kundendatenbank angeführt. In der Kunden-Variante hingegen erfolgt die Vorbefüllung aufgrund eines Logins. Die ausgeführten Operationen sind *nutzergruppenspezifisch*, da sie im jeweils anderen Nutzerkontext nicht sinnvoll sind (z.B. ein Kunde besitzt keine Kundendatenbank; ein Vertreter kann sich nicht als Kunde anmelden). Die im selben Beispiel dargestellte *systeminitiierte Aktion* der Vorbefüllung nach dem Login im Portalrahmen ist ein Beispiel für eine Abhängigkeit vom Technologiekontext. Da nicht alle Plattformen einen Login anbieten, erfordert auch hier die Laufzeitumgebung ein geändertes Verhalten.

Eine Herausforderung stellt zudem der **variierende Umfang der Daten** dar. Inhaltliche Varianten unterscheiden sich im Umfang der verarbeiteten Informationen und damit in den Daten, die im Anwendungskontext verfügbar sind. Da sich das Verhalten auf Elemente aus dem Anwendungskontext bezieht - sei es als Ein-/Ausgabeparameter oder als Auslöser - ist es daher möglich, dass die angeführten Operationen einer Variante nicht unverändert auf eine weitere Variante anwendbar sind. Referenzierte Elemente sind hier ggf. nicht existent.

Bei der manuellen Erstellung einer Variante passt der Entwickler das Verhalten entsprechend dem Nutzungs- und Technologiekontext an. Hierbei werden ggf. **Operationen durch Alternativen** ersetzt, die in der Variante verfügbare Daten und Funktionalitäten der technischen Umgebung verwenden.

Anforderungen an die Modellierung

Tabelle 5.3 fasst die beobachteten Eigenschaften dialogbasierter Benutzungsschnittstellen hinsichtlich des Verhaltens zusammen. Unter (IR.10)-(IR.16) sind die verhaltensrelevanten Eigenschaften aufgeführt. Insbesondere sind dies die *Sicht- und Editierbarkeit von Informationen* (IR.10), die *Validierung von Daten* (IR.11), die *Reaktion von Elementen* auf Änderung des Anwendungskontexts (IR.12), sowie *Aktionen*, die von Elementen (IR.13) oder explizit vom Nutzer bzw. der Laufzeitumgebung ausgelöst werden (IR.14).

Die erforderlichen Informationen zur Modellierung sind in den folgenden Spalten aufgeführt. Die Spalte **Elementzuordnung** zeigt an, welchem Elementtyp (*Gruppe, Element*) ein entsprechendes Verhalten zugeordnet werden kann. *Global* zeigt an, dass das Verhalten nicht einem Element zugeordnet sein muss, sondern übergreifend für die Anwendung angegeben werden kann (z.B. bei *systeminitiierten Aktionen*). Der **Auslöser** für ein Verhalten kann das veränderte Element selbst (*Element*), eine Änderung von Daten im Anwendungskontext (*Kontext*) oder ein Ereignis von außen sein, das durch den Nutzer oder das System angestoßen wird (*Extern*). Die Spalte **Eingabe** beschreibt die Art der Eingabeparameter für das Verhalten. Hierbei können entweder der Wert des Elements selbst oder Daten aus dem Anwendungskontext (*Kontext*) relevant sein. Die Spalte **Operation** beschreibt die Art der angestoßenen Aktion. Hier ist vermerkt, ob es sich um ggf. komplexe Programmlogik handelt (z.B. den Aufruf einer Operation im Anwendungskontext) oder das Verhalten über die Angabe von Regeln erfolgen kann (*regelbasiert, patternbasiert*). In der Spalte **Wirkung auf** ist beschrieben, welche Art von Auswirkung die Operation hat. Die Operation kann einerseits auf das Element selbst (z.B. dessen *Sichtbarkeit*, *Validität*, *Restriktionen*, *Inhalt*) anderseits auf Daten im Anwendungskontext wirken.

ID	Eigenschaften der Benutzungsschnittstelle	Anforderungen an die Modellierung				
Modellierung des Verhaltens						
		Elementzuordnung	***Auslöser***	***Eingabe***	***Operation***	***Wirkung auf***
(IR.10)	**Sicht-/Editierbarkeit von Informationen**	Datenelement, Gruppe	Kontext	Kontext	Regelbasiert Programmlogik	Element (Sichtbarkeit)
(IR.11)	**Validierung eingegebener Daten**	Datenelement, Gruppe	Element	Element Kontext	Patternbasiert Programmlogik	Element (Validität)
(IR.12)	**Reaktionen auf Änderungen im Kontext**	Datenelement	Kontext	Kontext	Programmlogik	Element (Inhalt, Restriktionen)
(IR.13)	**Elementinitiierte Aktionen**	Datenelement, Gruppe	Element	Kontext	Programmlogik	Kontext
(IR.14)	**Nutzer- und Systeminitiierte Aktionen**	Global,Daten-element, Gruppe	Extern	Kontext	Programmlogik	Kontext
Technische Varianten						
(IR.15)	**Verhaltensänderung aufgrund alternativem Systemkontext**	*ändern/hinzufügen entfernen*	-	*modifizierter Kontext*	*modifizierte Programmlogik*	*modifizierten Kontext*
Inhaltliche Varianten						
(IR.16)	**Verhaltensänderung aufgrund alternativem Nutzer- bzw. Anwendungskontext**	*ändern/hinzufügen entfernen*	*modifizierter Kontext*	*modifizierter Kontext*	*modifizierte Regeln, Patterns, Programmlogik*	*modifizierter Kontext*

Tabelle 5.3. Anforderungen und Informationsbedarf (Verhalten)

Die Anforderungen, die sich zur Unterstützung von technischen und inhaltlichen Varianten ergeben sind in **(IR.15)** und **(IR.16)** dargestellt. In beiden Fällen muss *Verhalten geändert, hinzugefügt oder entfernt* werden können. Zur Beschreibung einer **technischen Variante** müssen Informationen vorhanden sein, wie sich Eingabeparameter, Programmlogik sowie die Wirkung eines Verhaltens in einer Variante verändern. Bei **inhaltlichen Varianten** können sich durch den Wegfall bzw. die Hinzunahme von Datenelementen in der Anwendung zudem die Auslöser verändern. Da Datenelemente auch in Regeln referenziert werden, müssen diese ebenfalls angepasst werden können.

5.6 Nicht-funktionale Eigenschaften

Neben den genannten funktionalen Eigenschaften (Struktur, die Ein-/Ausgabe und das Verhalten) weisen Benutzungsschnittstellen Merkmale auf, die keinen direkten Einfluss auf die Funktionsweise der Benutzungsschnittstelle haben und dennoch maßgeblich deren Nutzbarkeit beeinflussen. Insbesondere sind dies (vgl. z.B. [35, 150, 222]):

- Gruppen- und Feldbezeichnungen
- Erläuterungs- und Hilfetexte
- Fehlermeldungen
- Piktogramme (Icons)

Gruppen- und Feldbezeichnungen beschreiben für jedes Dialogelement (Gruppen, Interaktionselemente) den von ihm repräsentierten Inhalt. **Erläuterungstexte** dienen der Erklärung der Fragen bzw. Fragegruppen und erläutern den Kontext. Erläuterungstexte sind dabei immer sichtbar und daher ein integraler Bestandteil der Benutzungsschnittstelle. **Hilfetexte**

sind optional angezeigte Informationen, die zu einer Frage bzw. einem Frageblock vom Benutzer angefordert werden können. **Fehlermeldungen** zeigen fehlerhafte Eingaben an und geben Hinweise zur Vermeidung. **Piktogramme** werden genutzt, um die Erkennbarkeit von Elemente der Benutzungsschnittstelle zu erhöhen.

Diese Elemente sind in hohem Maße nutzergruppen- und plattformspezifisch und tragen zur Nutzbarkeit und Verständlichkeit für den Nutzer bei. Sie unterscheiden sich z.T. stark in Anwendungsvarianten. Gruppen und Feldbezeichnungen sowie Erläuterungstexte variieren abhängig von der angesprochenen Nutzergruppe und unterscheiden sich dabei in Detaillierung und verwendeter Sprachlichkeit. Zudem sind sie den technischen Restriktionen unterworfen, die Interaktions- und Technologiekontexte vorgegeben.

Insbesondere *Erläuterungstexte, Hilfen und Fehlermeldungen* unterscheiden sich signifikant hinsichtlich der fachlichen und technischen Anforderungen. Sie können aus fachlicher Sicht z.B. für *Vertreter* knapper ausfallen als für *Endkunden*, da von einem größeren Fachwissen ausgegangen werden kann. Aus technischer Sicht hängt die Länge zudem von Geräterestriktionen ab, sodass auf mobilen, kleinen Geräten weniger Raum zur Erläuterung verfügbar ist. Der technische Kontext bestimmt zudem, ob ggf. alternative Texte für besondere Nutzungen notwendig sind, z.B. *aria-labels* für *Screenreader* oder alternative Fragetexte bei Wiederholungen in sprachbasierten Benutzungsschnittstellen. Bei der Verwendung von Piktogrammen kommen zudem kulturelle Unterschiede hinzu, welche die Auswahl maßgeblich bestimmen (vgl. [152]).

Anforderungen an die Modellierung

Tabelle 5.4 enthält die Anforderung, die sich zur Unterstützung der nicht-funktionalen Eigenschaften einer Benutzungsschnittstelle ergeben **(IR.17)**. Anwendungsvarianten unterscheiden sich abhängig vom Nutzungs-, Interaktions- und Technologiekontext in der Darstellung von Textelementen und Abbildungen. Die resultierende Anforderung besteht darin, dass abhängig vom Kontext unterschiedliche **Materialien** (*Ressourcen*, *Assets*) angegeben werden können, die in Varianten zu verwenden sind.

ID	Eigenschaften der Benutzungsschnittstelle	Anforderungen an die Modellierung
Inhaltliche und technische Varianten		
(IR.17)	Verwendung spezifischer Ressourcen für Gruppen und Datenelemente abhängig von ■ Technologie-/Interaktionskontext ■ Nutzungskontext (z.B. Nutzergruppe, Land/Sprache)	**Variable Materialien für Interaktionselemente** ■ **Gruppen-/Feldbezeichnungen, Erläuterunstexte, Hilfetexte, etc.** ■ **Fehlermeldungen, Hinweise, etc.** ■ **Piktogramme, Abbildungen, …**

Tabelle 5.4. Anforderungen und erforderliche Informationen (nicht-funktional)

Da die erforderlichen Assets maßgeblich vom Interaktions- und Technologiekontext abhängig sind, sind in Tabelle 5.4 exemplarisch Materialien aufgeführt, die üblicherweise in grafischen Benutzungsschnittstellen verwendet werden.

5.7 Diskussion der Ergebnisse

Der in Kapitel 3 vorgeschlagenen datenzentrierte Ansatz setzt voraus, dass Struktur und Aufbau, Interaktionselemente zur Ein-/Ausgabe und das Verhalten technologieneutral beschrie-

ben werden können. Das Ziel der Analyse war die Feststellung, welche Eigenschaften dialogbasierte Benutzungsschnittstellen aufweisen, welche Informationen zur deren Generierung erforderlich sind und welche Anforderungen sich daraus an die Modellierung ableiten. Die mit der Analyse verbundenen Forschungsfragen unter **(FF.1)** bestanden darin, welche Informationen zur Generierung grundsätzlich und für inhaltliche und technische Varianten benötigt werden. Zudem sollte festgestellt werden, ob diese Informationen technologieneutral formuliert werden können, um sie auf multiple Interaktions- und Technologiekontexte anzuwenden. Die resultierenden Anforderungen sind in den Tabellen 5.1 bis 5.4 zusammengefasst.

Ergebnisse: Die Anforderungen konnten so formuliert werden, dass sie keinen Bezug zu einer spezifischen Technologie aufweisen. Die zur Modellierung erforderlichen Informationen beschreiben ausschließlich Eigenschaften der Daten, deren Semantik und deren Beziehungen untereinander. Dies deutet darauf hin, dass sie auf unterschiedliche Interaktions- und Nutzungskontexte anwendbar sind. Auch die zur Bildung technischer und inhaltlicher Varianten benötigten Informationen (IR.3-IR.4, IR.8-IR.9, IR.15-IR.16 und IR.17) sind technologieneutral. Die Herleitung einer Variante hängt zwar von einem spezifischen Kontext ab (z.B. in (IR.8), *Regeln zur Auswahl von Interaktionselementen*), basieren jedoch ebenfalls ausschließlich auf Eigenschaften der Daten.

Diese Ergebnisse zeigen, dass dialogbasierte Benutzungsschnittstellen und deren Varianten technologieneutral beschrieben werden können. Sie stützen die These der Arbeit, dass ein technologieneutrales Modell zur automatischen Erzeugung dialogbasierter Benutzungsschnittstellen genutzt werden kann.

Potentielle Einschränkungen und offene Fragestellungen: Der Nachteil des zur Analyse angewendeten top-down-Ansatzes liegt in der eingeschränkten Sicht auf die Menge der untersuchten Anwendungen, die auf einer beschränkten Menge an Technologien beruht. Aus diesem Grund kann kein Anspruch auf Vollständigkeit der identifizierten Informationen abgeleitet werden. Zukünftige Untersuchungen insbesondere neuer Technologien können zu Erweiterungen der Anforderungen führen, die sich nicht aus den bisher untersuchten Anwendungen ableiten ließen.

Die Analyse fokussierte auf grafische Benutzungsschnittstellen. Alternative Interaktionsformen wurden im Rahmen der Arbeit nicht systematisch untersucht. Es erfolgten lediglich informale Vergleiche, speziell im Bereich der sprachbasierten Eingabe, bei denen die Anwendbarkeit der identifizierten Anforderungen betrachtet wurde. Diese ergaben, dass bestehende Ansätze (z.B. Konversationen über Google Assistant mit DialogFlow [80, 81] oder VoiceXML [137] modellierte Dialogabläufe) prinzipiell dieselben Restriktionen wie grafische Benutzungsschnittstellen für restriktive Technologien (z.B. *mobile Anwendungen*) aufweisen.

6. Modellierung von Benutzungsschnittstellen und Varianten

Dieses Kapitel beschreibt einen neuartigen Ansatz zur datenzentrierten Modellierung von Benutzungsschnittstellenvarianten dialogbasierter Anwendungen. Er verwendet ein technologieneutrales, um semantische Informationen erweitertes Modell der verarbeiteten Daten zur Beschreibung. Die Definition von Anwendungsvarianten erfolgt basierend auf diesem Modell durch Beschreibung inhaltlicher Unterschiede zwischen den Varianten.

Beiträge des Kapitels. Der Ansatz leistet einen Beitrag zur modellgetriebenen Entwicklung von Anwendungsvarianten. Die Beiträge sind im Einzelnen:

- ein Modellierungsansatz zur Beschreibung von Anwendungen durch die Informationen angereichertes Modell der verarbeiteten Daten
- eine Methode zur redundanzfreien Definition von Varianten
- eine Beschreibungssprache zur datenzentrierten Modellierung von Benutzungsschnittstellen sowie zur Definition von Varianten

6.1 Zielsetzung

Das Ziel dieses Kapitels ist die Darstellung der konkreten Modellierung für dialogbasierte Anwendungen und deren Varianten. Es beantwortet die in Abschnitt 1.3 unter **(FF.2)** formulierte Forschungsfrage, wie die zur vollständigen Generierung dialogbasierter Anwendungen notwendigen Informationen in einem Modell abgebildet und darüber hinaus zur Beschreibung von Varianten genutzt werden können.

Die an die Modellierung gestellten Anforderungen wurden in Abschnitt 3.1, Tabelle 3.1 unter **(A.1)** zusammengefasst. Die Modellierung muss danach hinsichtlich der enthaltenen Informationen vollständig **(A1.1)** sowie zur Erzeugung technischer Varianten **(A1.2)** und inhaltlicher Varianten **(A1.3)** geeignet sein. In Kapitel 5 wurden die hierzu erforderlichen Informationen identifiziert (vgl. Tabellen 5.1-5.3, (IR.1)-(IR.16)).

Zur Beantwortung der Forschungsfrage wird als Ergebnis eine Modellierung dargestellt, welche die o.g. Anforderungen erfüllt und die erforderlichen Informationen enthält. Es wird darauf aufbauend eine Methodik zur Spezifikation von Varianten angegeben und abschließend eine Beschreibungssprache für die Modellierung dargestellt.

6.2 Lösungsansatz

In Abschnitt 3.3 wurde der Lösungsansatz zur Modellierung dialogbasierter Anwendungen im Überblick skizziert. Er besteht darin, **ein einziges, zentrales Anwendungsmodell** zu verwenden, das die zur Generierung *technischer Varianten* notwendigen Informationen enthält. Die Informationen sind unabhängig von einer konkreten Zieltechnologie formuliert und bilden ein datenzentriertes Modell der Anwendung.

Das Anwendungsmodell dient als Grundlage zur Spezifikation *inhaltlicher Varianten* mittels (optionaler) **Variantenbeschreibungen**. Der Lösungsansatz besteht darin, die Unterschiede hinsichtlich Inhalt und Verhalten einer Variante zum Anwendungsmodell zu beschreiben.

Diese werden als Modifikationen auf ein Anwendungsmodell angewendet und führen zu einem modifizierten Anwendungsmodell.

Das Anwendungsmodell und die Variantenbeschreibung werden im Folgenden als **Metamodelle**[37] beschrieben. Die Nutzung von Metamodellen ermöglicht es, multiple Repräsentationen der Modelle zu erstellen und damit in verschiedenen technischen Umgebungen anwendbar zu sein.

Basierend auf den Metamodellen wird abschließend eine **konkrete Beschreibungssprache** in Form einer domänenspezifischen Sprache (*Domain Specific Language*, *DSL*) sowohl für das Anwendungsmodell als auch die Variantenbeschreibung vorgestellt.

Folgende Ergebnisse werden im Verlauf des Kapitels vorgestellt:

- Metamodell zur Beschreibung dialogbasierter Anwendungen (Abschnitt 6.3)
- Metamodell zur differentiellen Beschreibung inhaltlicher Varianten (Abschnitt 6.4)
- Repräsentation als *Domain Specific Language* (Abschnitt 6.5)

Abschnitt 6.6 fasst die Ergebnisse zusammen und diskutiert den vorgeschlagenen Ansatz in Bezug zu den in Abschnitt 5 formulierten Anforderungen.

6.3 Metamodell zur Beschreibung dialogbasierter Anwendungen

Die Entwicklung des **Metamodells für dialogbasierte Benutzungschnittstellen** erfolgte iterativ entlang der in den Abschnitten 5.3 - 5.5 identifizierten Themenbereiche und Anforderungen. Hierzu wurden die Informationen zu

- *Struktur und Aufbau* (IR.1, IR.2),
- *Typisierter Ein-/Ausgabe* (IR.5) und
- *Verhalten* (IR.10-IR.14)

im Metamodell beschrieben und sukzessive um Informationen zur Semantik der modellierten Elemente ergänzt (IR.6, IR.7).

Im Folgenden wird das Metamodell entlang dieser Themenbereiche dargestellt. Abschnitt 6.3.1 beschreibt die Modellierung der *Struktur und des Aufbaus*. In Abschnitt 6.3.2 wird die Umsetzung der Anforderungen zur *typisierten Ein-/Ausgabe* im Modell dargestellt. Da zur Beschreibung des Verhaltens sowohl Daten als auch Operationen aus dem Anwendungskontext genutzt werden, beschreibt Abschnitt 6.3.3 zunächst, wie *Abhängigkeiten zu Daten und Operationen* im Anwendungskontext modelliert werden. Darauf aufbauend wird die Modellierung des *Verhaltens* von Benutzungsschnittstellen dargestellt (Abschnitt 6.3.4). Abschnitt 6.3.5 fasst die Ergebnisse in einem Gesamtmodell zusammen und diskutiert die Lösung hinsichtlich der Anforderungen.

[37] Der Begriff *Metamodell* wird hier analog der Auffassung im MDD-Umfeld verwendet. Ein Metamodell ist dabei ein Modell, welches die Elemente und deren Beziehungen, Attribute und Eigenschaften einer Familie von Modellen beschreibt. Nach Mellor et al. [142] ist ein Metamodell ein Modell, welches das Modell einer Modellierungssprache beschreibt. Es definiert die Struktur, Semantik und Einschränkungen für eine Familie von Modellen. Nach Clark et al. [38] ist ein Metamodell ein Modell einer Sprache, das deren Elemente und Semantik beschreibt.

6.3.1 Struktur und Aufbau

Zur Ableitung der Benutzungsschnittstelle benötigt ein Generator Informationen zu deren Struktur und Aufbau. In Tabelle 5.1 wurden hierzu die Anforderungen zur hierarchischen Struktur, der inneren/äußeren Kohäsion und der temporalen Abfolge von Fragen unter (IR.1) und (IR.2) zusammengefasst.

Die hierarchische Natur von Benutzungsschnittstellen (IR.1) legt nahe, diese **über eine Baumstruktur zu beschreiben**. In der Literatur existieren Ansätze, die Baumstrukturen als Mittel zur Modellierung von Benutzungsschnittstellen verwenden. So verwendet Nichols [150] *m-wertige Bäume* zur Beschreibung der Struktur von Bedienungsoberflächen für technische Geräte (sog. *Appliances*, z.B. Bedienfelder für Kopierer und Videorecorder). Die Strukturbäume modellieren Gruppen und Eingabeelemente in einem hohen Detaillierungsgrad (z.B. durch feingranulare Bildung von Gruppen), die in einem Strukturbaum mit großer Tiefe resultiert. Nichols stellt fest, dass derart feingranular modellierte Baumstrukturen insbesondere zur Generierung von Oberflächen für gleichartige Geräte mit unterschiedlichen Restriktionen geeignet sind. Der von Nichols verwendete Ansatz konzentriert sich zwar auf die spezifische Anwendungskategorie der *Appliances*, kann aber grundsätzlich zur Beschreibung dialogbasierter Benutzungsschnittstellen erweitert werden.

Der in der Arbeit verfolgte Ansatz setzt darauf auf und nutzt als Grundlage **geordnete, m-wertige Bäume** (s. [42, 118, 206]), deren Knoten zur näheren Beschreibung von Knoteneigenschaften **attribuiert** werden können. Abbildung 6.1 zeigt das zugrundegelegte Prinzip.

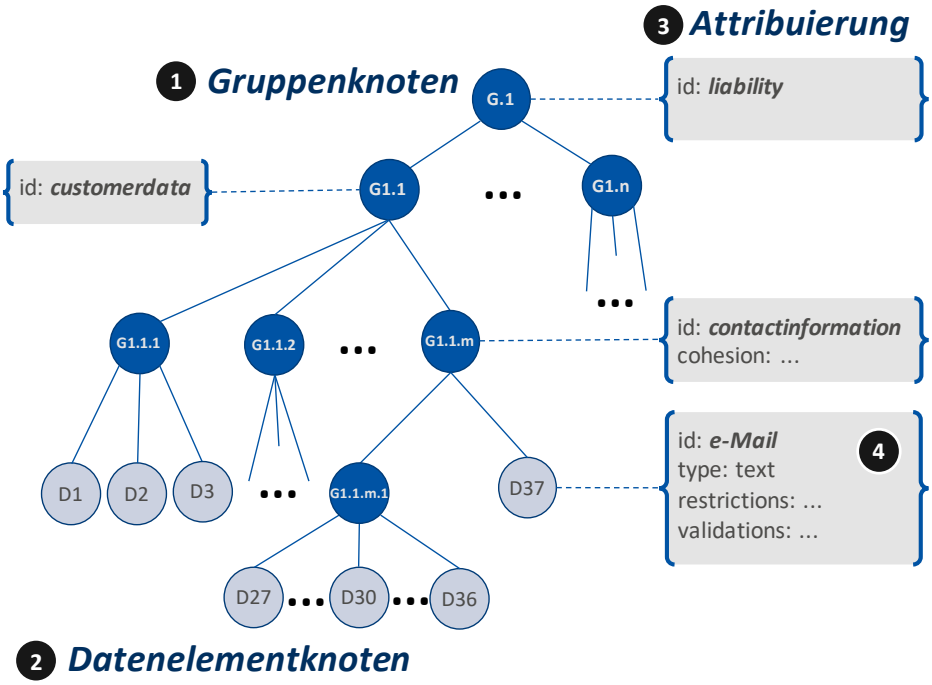


Abbildung 6.1. Strukturbaum einer Anwendung

Der Wurzelknoten des Baumes repräsentiert die Anwendungsbeschreibung, dem weitere Teilbäume oder Blätter zugeordnet werden. Die Knoten des Baums repräsentieren Gruppen (G_x, ①). Die Blätter repräsentieren die zu erfassenden Daten der Anwendung (D_x, ②). Die

Knoten auf jeder Hierarchiestufe des Baumes sind geordnet, wodurch sich eine Reihenfolge der Knoten des Baumes ableiten lässt.

Knoten und Blätter des Baums werden durch Attribute um Detailinformationen ergänzt ③. Darin wird z.B. verzeichnet, welche Gruppe ein innerer Knoten repräsentiert (z.B. *liability, customerdata, contactinfomration*) bzw. welcher Typ und welche Restriktionen für ein Datenelement gelten ④. Das Ergebnis ist ein **attribuierter Strukturbaum**, der die Daten der Anwendung strukturiert und die Informationen zur automatischen Generierung der Benutzungsschnittstellen trägt.

Modellierung der hierarchischen Struktur. Der dargestellte Strukturbaum dient als Basis zur Modellierung des Aufbaus und der Struktur im Metamodell. Abbildung 6.2 zeigt die Elemente des Metamodells als UML-Diagramm.

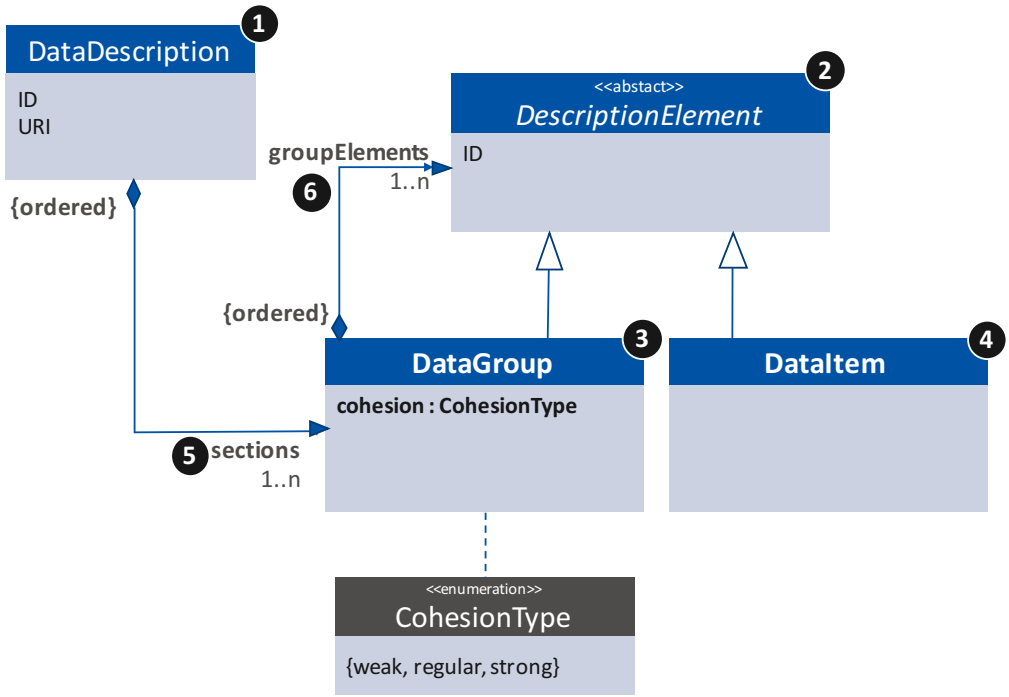


Abbildung 6.2. Abbildung der hierarchischen Struktur im Metamodell

Die Datenbeschreibung (*DataDecription* ①) repräsentiert den Wurzelknoten des Anwendungsmodells. Sie enthält eine Menge an Beschreibungselementen (*DescriptionElement* ②), die einerseits Gruppen (*DataGroups* ③), andererseits Datenelemente (*DataItems* ④) repräsentieren. Diese Elemente weisen gemeinsame Eigenschaften auf, die in der abstrakten Klasse *DerscriptionElement* zusammengefasst sind. Zur Modellierung der hierarchischen Struktur enthält die *DataDecription* eine geordnete Liste von *DataGroups* (*sections* ⑤). *DataGroups* bestehen wiederum aus einer einer geordneten Liste von *DerscriptionElements* (*groupElements* ⑥), in der *DataGroups* und *DataItems* enthalten sind.

Zur Erstellung eines Anwendungsmodells wird anhand der semantischen Zusammengehörigkeit der Daten ein Strukturbaum mit diesen Elementen aufgebaut. Die zu erfassenden Daten werden dabei als *DataItems* modelliert und aufgrund ihrer semantischen Zusammengehörigkeit in Gruppen und Untergruppen (*DataGroups*) zusammengefasst. Untergeordnete Gruppen und Daten werden als Unterknoten (*groupElement*) hinzugefügt. Durch die beschriebene

Modellierung wird die in (IR.1) geforderte hierarchische Struktur abgebildet. Eingabedaten werden zu sinnvollen Einheiten zusammengefasst und in einer hierarchischen Dialogstruktur beschrieben.

Modellierung der Kohäsion. Die **innere Kohäsion** der Daten (IR.1) ist bereits aus der hierarchischen Baumstruktur ablesbar. Durch die o.g. Modellierung entsteht eine Struktur von Knoten, deren Unterknoten gemäß ihrer Kohäsion zusammengefasst sind. Die Elemente innerhalb einer Gruppe weisen dabei die gleiche Kohäsion auf. Die Elemente in einer Untergruppe wiederum weisen untereinander eine stärkere Kohäsion auf als zu Elementen der umgebenden Gruppe. Die Kohäsion steigt damit auf jeder Ebene des Baumes.

Hieraus lässt sich jedoch nicht ablesen, ob Elemente eine **starke innere Kohäsion** (IR.1, Punkt 2) aufweisen, noch lässt sich die semantischen Bindung einer Gruppe zu ihrem Umfeld (**äußere Kohäsion** (IR.1), Punkt 3) ableiten. Diese Kohäsionsstärke wird im Metamodell über das Attribut *cohesion* beschrieben (s. Abbildung 6.2 ③). Die Kohäsionsstärke wird über die Werte *strong* bzw. *weak* spezifiziert. Eine mit *strong* angegebene Kohäsion bedeutet, dass die enthaltenen Elemente sehr eng miteinander verbunden sind und damit eine Einheit bilden (**starke innere Kohäsion**). *weak* bezeichnet eine gelockerte Kohäsion der enthaltenen Elemente zum Umfeld (**äußere Kohäsion**).

Mit dieser Modellierung ist die in (IR.1) geforderte Information zur *inneren und äußeren Kohäsion* der Daten und Gruppen im Modell verfügbar. Es lassen sich Gruppen beschreiben, die bei der Generierung in Formularbereichen auf einer Seite resultieren (*innere Kohäsion*). Eine besondere räumliche Nähe (*starke innere Kohäsion*) von Informationen kann abgeleitet und der Fragefluss für die Zielplattform umorganisiert werden (*äußere Kohäsion*).

Modellierung der Sequenz. Die Verwendung von *geordneten, m-wertigen Bäumen* zur strukturellen Beschreibung beinhaltet bereits die in (IR.2) geforderte Information zur Sequenz von Elementen. Die Modellierung der in Abbildung 6.2 dargestellten Beziehungen *sections* ⑤ und *groupElements* ⑥ als **geordnete Liste** (*{ordered}*) legt für Elemente die Reihenfolge fest. Hieraus ist bei der Generierung die Position einer Frage oder Gruppe im Fragefluss ableitbar (IR.2).

6.3.2 Typisierte Ein-/Ausgabe

In Abschnitt 5.4, Tabelle 5.2, wurden die erforderlichen Informationen zur Erzeugung der Interaktionselemente zur Ein-/Ausgabe für Datenelemente zusammengefasst. Insbesondere waren dies Informationen zum Typ und geltender Restriktionen zu einem Datenelement (IR.5, IR.6), aus welchem Standard-Interaktionselemente ableitbar sind. Zur Herleitung applikationsspezifischer Interaktionselemente sind darüber hinaus Informationen zur fachlichen Semantik eines Datums oder einer Gruppe (IR.6, IR.7) erforderlich.

Die Typeigenschaften eines Elements können als Attribute der Elemente des Strukturbaums aufgefasst werden. Im Metamodell werden sie als Attribute der Klasse *DescriptionElement* und deren Derivate (*DataItem, DataGroup*) modelliert.

Abbildung 6.3 stellt das um die typbezogenen Aspekte erweiterte Metamodell als UML-Diagramm dar. Typinformationen werden als Attribute ① und Restriktionen als assoziierte Objekte (*TypeRestriction* ②) modelliert. Die Beschreibung der fachlichen Semantik von Gruppen und Daten erfolgt über die Angabe von *SemanticTags* ③.

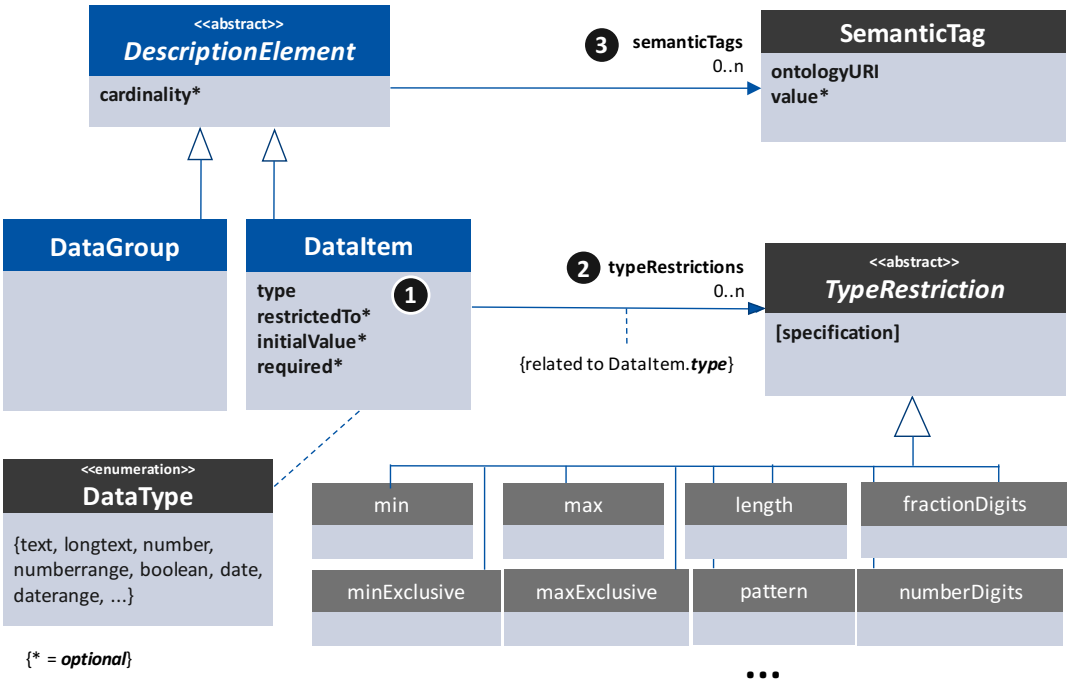


Abbildung 6.3. Modellierung der typisierten Ein-/Ausgabe

Modellierung der Typisierung. Zur Typisierung eines Datenelements wird der Klasse *DataItems* ein Attribut *type* hinzugefügt, welches den Basistyp des Elements festlegt (Abbildung 6.3 ①). In Tabelle 6.1 sind die im Metamodell vorgesehenen Datentypen zusammengefasst, die bei der Analyse beobachtet wurden (vgl. Abschnitt 5.4).

Deren Semantik basiert auf dem durch *XML-Schema* [61] definierten Typsystem[38] und Elementen der ISO/IEC 11404:2007 [106]. Hierbei sind insbesondere die atomaren, einfachen Datentypen (*atomic, primitive datatypes*[39]) relevant. Zur Vereinfachung der Erstellung des Anwendungsmodells wurden Datentypen zusammengefasst, die eine gleiche Repräsentation besitzen bzw. durch die Angabe von Restriktionen entsprechend spezialisiert werden können (z.B. Zusammenfassung von numerischen Werten wie *decimal, float, double* sowie deren Untertypen zu *number*). Für in der Analyse identifizierte Datentypen, die keine direkte Entsprechung zu einfachen Datentypen in *XML-Schema* haben, wurden weitere Typen ergänzt (*number-*, *date-* und *timerange*).

Modellierung der Spezifikation geltender Restriktionen. Restriktionen sind im Metamodell als *TypeRestriction*-Objekte modelliert (vgl. Abbildung 6.3), die mit *DataItems* assoziiert werden (*typeRestrictions* ②). Als Grundlage wurden die in XMLSchema definierten Restriktionen (*constraining facets*[40]) herangezogen. Diese wurden um Aspekte ergänzt, die zur Ableitung einer Benutzungsschnittstelle zusätzlich benötigt werden (z.B. Informationen zu Schrittweiten, Einheiten und Initialwerten). Tabelle 6.2 fasst die im Metamodell vorgese-

[38] https://www.w3.org/TR/xmlschema11-2/#typesystem

[39] https://www.w3.org/TR/xmlschema11-2/#built-in-primitive-datatypes

[40] https://www.w3.org/TR/xmlschema11-2/#rf-facets

Datentyp	Verwendung	XMLSchema*	Beispiel
text	Textueller Wert - z.B. Vorname, Nachname, Stadt	#string	`Max Mustermann`

longtext	Komplexer textueller Wert. Der enthaltene Text kann mehrzeilig sein und ggf. Elemente einer Auszeichnungssprache beinhalten (z.B. Formatierung über HTML)	#string	`Dies ist ein Langtext.`
			
			`Dieser kann sich über mehrere Zeilen erstrecken. Der Inhalt kann <b>formatiert</b> sein.`

number
- integer
- decimal
	Numerischer Wert. Die Form wird durch Restiktionen näher bestimmt, z.B. ob es sich um einen ganzzahligen oder rationalen Zahlenwert handelt. Zur Vereinfachung kann der Typ "integer" bzw. "decimal" angegeben werden, welche entsprechende Restriktionen implizieren.	#integer
		#decimal
		#float
		#double
			`2007`
			`10.2`
			`121.42883`
			`2E12`

numberrange	Eingabe eines Wertebereichs mit unterem und oberem Wert. Analog dem Datentyp number kann auch hier der Typ "integerrange" bzw. "decimalrange" angegeben werden.	---	`(200,410)` `(123.34,2343.43)`
boolean	Angabe eines Wahrheitswertes (Ja/Nein-Feld)	#boolean	`true` `false`

date	Angabe eines Datums im ISO-Format [ISO 8601]	#date
		#dateTime
			`2017-10-10+13:00`
			`2017-10-10`
			`2017-01-15T12:00:00`

day, month, year, yearMonth, monthDay	Fragmente eines Datums	#gDay, #gMonth, #gYear, #gYearMonth, #gMonthDay	`2017` `2017-12`
time	Angabe eines Zeitpunktes (Uhrzeit)	#time	`12:00:00`
duration	Angabe einer zeitlichen Dauer	#duration	`02:30:00`
daterange	Angabe eines Datumsbereichs	---	`(2017-10-10;2017-10-11)`
timerange	Angabe eines Zeitabschnitts	---	`(12:00:00;14:00:00)`
*** Dokumentation: https://www.w3.org/TR/xmlschema-2/# ...**			

Tabelle 6.1. Verwendete Datentypen für *DataItems*

hene Restriktionen in die Gruppen **Wertrestriktionen**, **Bereichsrestriktionen**, **Formatrestriktionen** und **Elementrestriktionen** zusammen.

Die **Wertrestriktionen** beschreiben die konkrete Beschränkung auf diskrete Werte, die das zu erfassende Datum annehmen kann. Diese Information kann in der Benutzungsschnittstelle zur Verwendung eines Interaktionselements genutzt werden, welches die Auswahl von vornherein auf eines oder mehrerer Werte aus der Wertemenge einschränkt (z.B. *1-aus-N* oder *M-aus-N-Auswahl*). Als Restriktion kann in der Wertemenge angegeben werden, auf welche die Eingabe beschränkt sein soll (*restrictedTo*). Zudem kann angegeben werden, ob ein oder mehrere Elemente gewählt werden können (*restrictionCardinality*).

Die **Bereichsrestriktionen** betreffen insbesondere numerische bzw. abzählbare Werte. Sie legen fest, ob eine Wertangabe beschränkt ist (*min, minExclusive*, *max, maxExclusive*) oder in einer festen Schrittweite (*step*) erfolgt (z.B. in Zehntelschritten bei Angabe von *step=0.1* für Dezimal-Werte in einem Bereich von *min=0.0* bis *max=1.0*). Zudem kann eine Einheit angegeben werden (*unit*).

Die **Formatrestriktionen** beinhalten Beschränkungen, die sich auf die Form der Werte beziehen. So kann für numerische Werte spezifiziert werden, ob es sich um einen ganzzahligen

Restriktion	Beschreibung	Anwendbarkeit	Beispiel
Wertrestriktionen			
restrictedTo	Einschränkung auf eine feste Wertemenge des angegebenen Typs	alle*	{"Hund"|"Katze"|"Maus"} {"100€"|"250€"|"500€"}
restrictionCardinality	Multiplizität der Auswahl. Standard: 1	alle*	{0}, {1}, {*}, {0..n}, {1..7}
Bereichsrestriktionen			
min	minimaler Wert (#wert>=min)	number, date, time	30, 2017-08-08, 00:00:15
max	maximaler Wert (#wert<=max)	number, date, time	30, 2017-08-08, 00:00:15
minExclusive	minimaler Wert (#wert>minExclusive)	number, date, time	30, 2017-08-08, 00:00:15
maxExclusive	maximaler Wert (#wert<manExclusive)	number, date, time	30, 2017-08-08, 00:00:15
step	Schrittweite	number, date, time	1, 0.5, 00:15:00
unit	Mengeneinheit	alle*	"m","cm","kg","€"
Formatrestriktionen			
fractionDigits	Anzahl Nachkommastellen	number	2 (z.B. für Geldbeträge)
numberDigits	Anzahl Vorkommastellen	number	5 (z.B. für Postleitzahlen)
length, minLength, maxLe	Anzahl an Stellen/Zeichen	alle* - außer date, time und Derivate	300 (z.B. für Freitexteingabe)
pattern	Formatbeschränkung (Regulärer Asudruck)	alle* - außer date, time und Derivate	[a-Z1-9\.-]@[a-Z1-9\.-]\.[a-Z]{2,3} (z.B. für E-Mail)
Elementrestriktionen			
required	Indikator für ein Pflichtfeld		true
initialValue	initialer Wert des Feldes (abhängig vom Typ)	alle	12, "Hund", 1.2, 12:00:00, true, false
outputOnly	Ausgabefeld (keine Eingabe möglich)	alle	true, false
hidden	nicht sichtbares Feld	alle	true,false
* Ausgenommen: boolean			

Tabelle 6.2. Verwendete Restriktionen für Datenelemente

oder Dezimalwert handelt und wie viele Stellen vor bzw. nach dem Komma gestattet sind. Für bestimmte Typen kann die Länge der Eingabe beschränkt und ein regulärer Ausdruck (*pattern*) bestimmt werden, der das erwartete Format festlegt.

Da es sich bei *XMLSchema* um ein System zur Beschreibung von Datentypen handelt, sind naturgemäß keine Informationen enthalten, die die Laufzeit einer potentiellen Benutzungsschnittstelle betreffen. Unter **Elementrestriktionen** sind ergänzende Restriktionen aufgeführt, die beim Aufbau der Benutzungsschnittstelle relevant sind. So kann für ein Eingabefeld bestimmt werden, ob es sich um ein Pflichtfeld handelt (*required*) und ein Wert (*initialValue*) zugewiesen werden, der als Vorbelegung verwendet wird. Zudem kann ein Feld als reines Ausgabefeld (*outputOnly*) oder als unsichtbar (*hidden*) gekennzeichnet werden.

Einige der dargestellten Restriktionen sind **nur auf bestimmte Typen anwendbar**. Tabelle 6.2 stellt in der Spalte *Anwendbarkeit* dar, auf welche Datentypen eine Restriktion anwendbar ist. Im UML-Diagramm des Metamodells ist dieser Aspekt als Einschränkung (*{related to type}*) an der Relation zu den Restriktionen vermerkt (vgl. Abbildung 6.3 ②).

Modellierung der fachlichen Semantik von Daten und Gruppen. Die Umsetzung der Anforderungen (IR.6), (IR.7) und (IR.8) stellt eine Herausforderung dar. Elemente des Anwendungsmodells mit einer spezifischen Semantik zu versehen, schränkt dessen Nutzbarkeit auf bestimmte Anwendungsdomänen ein. Wäre die konkrete Semantik von Gruppen und

Datenelementen fester Bestandteil der Metamodellspezifikation, wäre dieses nicht für beliebige Anwendungsdomänen nutzbar. Die Modellierung der Semantik muss daher zusätzliche Kriterien erfüllen. Sie muss **spezifisch** sein um (IR.6) und (IR.7) zu erfüllen. Generatoren müssen die Semantik von Elementen des Modells erkennen, damit entsprechende domänenspezifische Interaktionselemente hergeleitet werden können. Sie muss jedoch auch **generell** sein, damit Anwendungsmodelle von beliebigen Generatoren interpretiert werden können, selbst wenn diese eine Anwendungsdomäne nicht kennen (IR.8). Dies erfordert eine Modellierung, bei der die Semantik als zusätzliche Information erfolgt und deren Spezifikation außerhalb des Anwendungsmodells liegt.

Zur Beschreibung der Semantik werden **semantische Annotationen**[41] verwendet [116, 159]. Dieser Ansatz stammt aus dem Semantic Web Umfeld. Die Idee besteht darin, bestehende Entitäten (Dokumente, Inhalte, Strukturen, etc.) um maschinell auswertbare Informationen zu ihrer Bedeutung anzureichern und damit die Grundlage zu ihrer Interpretation zu schaffen. Zur Spezifikation der Semantik dienen dabei Ontologien. Eine Ontologie ist in diesem Zusammenhang als Wissensbasis zu verstehen, welche Wissen zu einer Anwendungsdomäne modelliert [95]. Insbesondere werden dort die Bedeutung, Eigenschaften und Zusammenhänge von Entitäten der Domäne beschrieben. Über eine semantische Annotation wird die Referenz auf die Entität einer Ontologie beschrieben. Für das annotierte Element wird damit bestimmt, dass es sich dabei um ein Element mit den in der Ontologie beschriebenen Eigenschaften handelt.[42]

Die **Semantik einer Gruppe oder eines Datenelements** kann als weiteres Attribut eines Strukturbaum-Knotens aufgefasst werden. Ein Knoten wird mit semantischen Annotationen (*SemanticTags*) versehen, welche die grundlegenden Typeigenschaften des Knotens um darüber hinausgehende Semantik erweitern. In Abbildung 6.4 ist die Umsetzung im Metamodell dargestellt. Einem *DescriptionElement* können beliebig viele *SemanticTags* ① zugewiesen werden, die als Referenz auf eine Entität einer externen Ontologie spezifiziert werden ②. Die referenzierte Ontologie ist dabei nicht Teil des Metamodells.

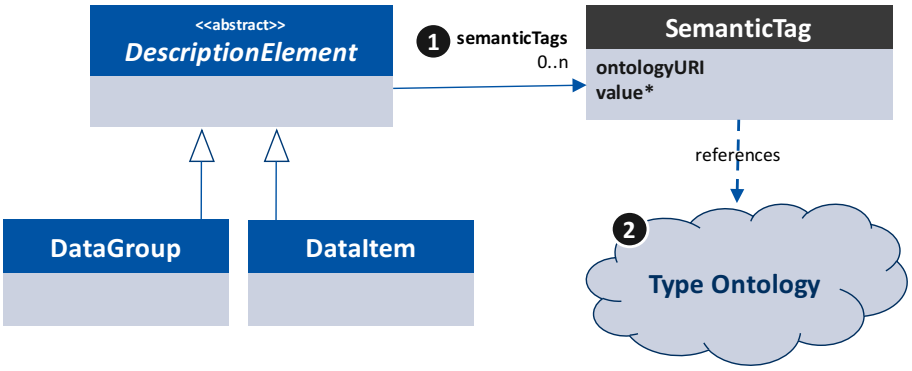


Abbildung 6.4. SemanticTags zur näheren Bestimmung von Gruppen und Datenelementen

[41] http://ontotext.com/knowledgehub/fundamentals/semantic-annotation

[42] Ein ähnlicher Ansatz sind beispielsweise *Annotations* in der Programmiersprache Java [83]. Hier dienen Annotationen zur Anreicherung von Klassen, Attributen und Methoden mit *Metadaten*. *Prozessoren* können diese Elemente über deren *erweiterten Semantik* identifizieren und verarbeiten (z.B. durch Anpassung des Verhaltens einer Klasse).

Das UML-Instanzdiagramm in Abbildung 6.5 zeigt die Nutzung semantischer Tags für *DataItems* am Beispiel zur Erfassung einer IBAN sowie erkrankter Körperbereiche aus Abschnitt 6.3.2. Das Datenelement für eine IBAN ① besitzt den Grundtyp *text* und wird mit einer semantischen Annotation *iban* ② erweitert, welche in einer Ontologie *Extended Types Ontology* ③ definiert wurde. Die Ontologie ist eindeutig mittels einer URI referenziert (*http://mimesis.solutions/types/ext/v1#*) und definiert eine Reihe erweiterter Eingabetypen, die domänenübergreifend genutzt werden können.

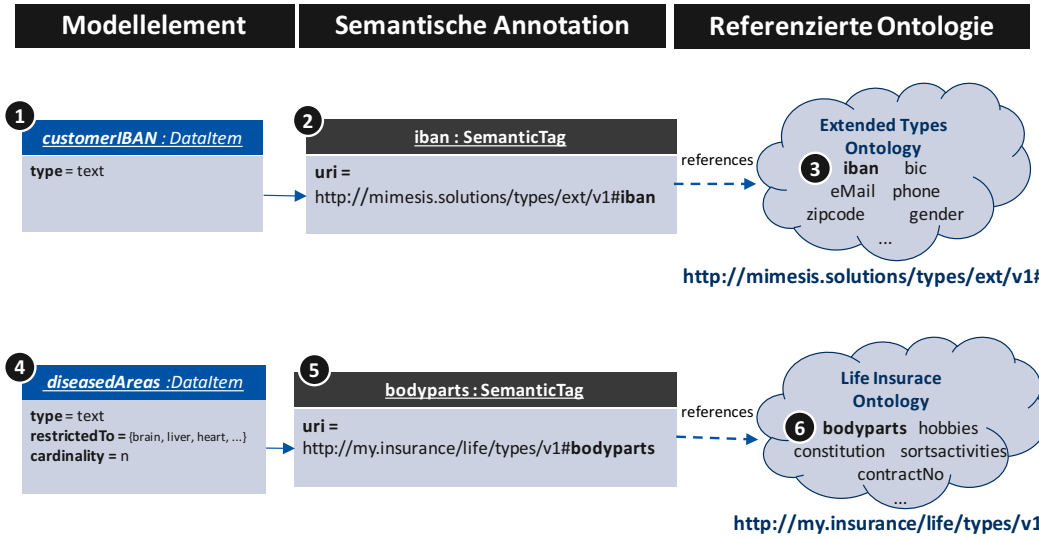


Abbildung 6.5. SemanticTags für IBAN und Körperbereiche

Analog erfolgt die semantische Beschreibung der Erfassung erkrankter Körperbereiche (*diseasedAreas* ④). Das Datenelement ist über die Restriktionen als *M-aus-N-Auswahl* beschrieben. Es wird durch die semantische Annotation *bodyparts* ⑤ fachlich näher bestimmt. Die Semantik von *bodyParts* ist in einer domänenspezifischen Ontologie für Lebensversicherungs-Anwendungen beschrieben (*Life Insurance Ontology* ⑥), welche durch die URI *http://my.insurance/life/types/v1#* eindeutig referenziert wird.

Semantische Annotationen für *DataGroups* beschreiben die Semantik darin enthaltener Daten als Einheit. Abbildung 6.6 zeigt dies für die Erfassung der *Konstitution* einer Person (*constitution* ①). Sie enthält Datenelemente zur Eingabe von Geschlecht (*gender*), Körpergröße (*height*) und Gewicht (*weight*) ②, die jeweils durch semantische Annotationen ③ semantisch bestimmt sind.

Die Gruppe *constitution* ① ist durch die semantische Annotation *constitution* ③ näher beschrieben. Diese ist in einer *Life Insurance Ontology* ④ spezifiziert. Dort ist festgelegt, dass sich die Konstitution aus Elementen mit der Semantik *gender*, *height* und *weight* zusammensetzt. Mittels dieser Information kann ein spezifischer Generator untersuchen, ob die erforderlichen Datenelemente in der konkret modellierten Gruppe enthalten sind und diese entsprechend zusammengefasst in einem Interaktionselement darstellen (IR.7).

6.3.3 Abhängigkeiten zum Anwendungskontext

In Abschnitt 5.2 wurde dargestellt, dass zur Ausführung einer Anwendung ein Anwendungskontext erforderlich ist, der einerseits den Zustand der Daten enthält, andererseits die Opera-

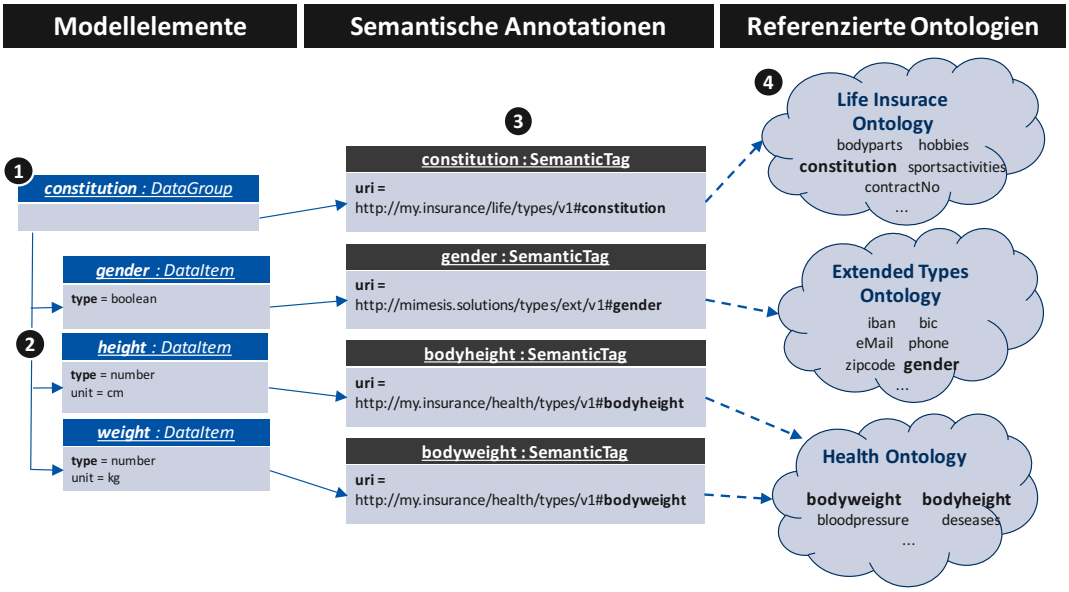


Abbildung 6.6. SemanticTags zur näheren Bestimmung einer Gruppe

tionen zur Umsetzung des Verhaltens zur Verfügung stellt. Hierzu muss in der Benutzungsschnittstelle angegeben werden, mit welchen Daten Interaktionselemente assoziiert sind und welche Operationen zur Umsetzung des Verhaltens genutzt werden.

Die Voraussetzung für die Modellierung des Verhaltens ist, dass Operationen beschrieben werden können, deren konkrete Implementierungen zur Laufzeit im Anwendungskontext vorliegen. Zudem muss beschrieben werden, welche Daten im Anwendungskontext bei der Ausführung beteiligt sind. Hierzu sind

- Referenzen auf Daten innerhalb des Anwendungsmodells
- Referenzen auf Operationen

erforderlich, über die bei der Erzeugung einer Benutzungsschnittstelle auf konkrete Operations-Implementierungen und Daten im Anwendungskontext zur Laufzeit geschlossen werden kann.

Im Folgenden wird der grundsätzliche Ansatz zur Modellierung von Referenzen auf Daten und Operationen des Anwendungsmodells beschrieben. Er bildet die Grundlage zur konkreten Modellierung des Verhaltens, welche in Abschnitt 6.3.4 dargestellt wird.

Referenzierung von Daten innerhalb des Anwendungsmodells

Zur Referenzierung von Daten innerhalb des Anwendungsmodells können die Baumeigenschaften des Anwendungsmodells ausgenutzt werden. Die Modellierung der von der Anwendung verarbeiteten Daten als Strukturbaum impliziert bereits eine Struktur, in welcher einzelne Knoten identifiziert werden können. Aufgrund der Baumeigenschaften kann jedes Datum im Strukturbaum eindeutig über einen Pfad ausgehend vom Wurzelknoten zum betroffenen Unterknoten angegeben werden. Abbildung 6.7 zeigt dies am Beispiel des Strukturbaums für die Erfassung der Kundendaten. Es zeigt die Referenzierung des Datenelements *zip* als Pfad innerhalb des Strukturbaums. Ausgehend vom Wurzelknoten (*liability*) wird diese als

Pfad angegeben, der über die Knoten *customerdata*, *contactinformation* und *address* bei *zip* endet.

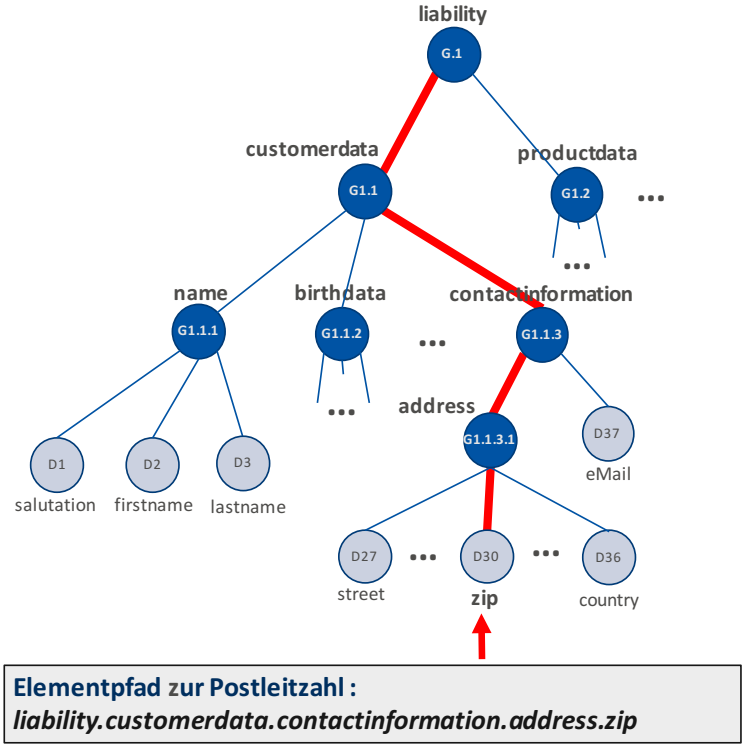


Abbildung 6.7. Identifikation von Elementen im Strukturbaum einer Anwendung

Als Notation zur Beschreibung des Pfades kann eine einfache Punkt-Notation (*dotted notation*) verwendet werden. Diese wird beispielsweise auch in JSF [158] oder JavaScript für Pfadangaben zur Navigation durch Datenstrukturen verwendet. Da alle Knoten im Strukturbaum (*DescriptionElements*) eine eindeutige Bezeichnung (*ID*) besitzen, kann der Pfad als Verkettung der IDs im Baum angegeben werden. Ein Elementpfad lässt sich mit folgendem regulären Ausdruck beschreiben:

$$\text{elementpfad} = \text{<Wurzelknoten-ID>} \ (\text{“.”} \ \text{<Baumknoten-ID>})^* \ \text{“.”} \ \text{<Zielknoten-ID>}$$

Ausgehend von der ID des Wurzelknotens (*Wurzelknoten-ID*) werden durch Punkte getrennt die auf dem Pfad zum Ziel liegenden Knoten (*Baumknoten-IDs*) aneinandergereiht (vgl. Abbildung 6.7, unten).

Mit dieser Notation sind alle Elemente des Anwendungsmodells identifizierbar und können an anderen Stellen im Modell (z.B. zur Angabe als Parameter einer Operation) eindeutig referenziert werden.

Beschreibung von Operationen und deren Parameter

Zur Beschreibung des Verhaltens muss im Anwendungsmodell spezifiziert werden, welche Funktionalität zur Umsetzung aufgerufen werden und welche Ein- und Ausgabeparameter die Operation verwenden soll. Aus Sicht des Anwendungsmodells kann dabei von der konkreten Implementierung der Operationen abstrahiert werden. Für die Benutzungsschnittstelle

ist lediglich relevant, dass eine *Operation mit einem erwarteten Verhalten* aufgerufen wird. Es muss eine Operation ausgeführt werden, die eine *definierte Semantik* besitzt.

Analog zur Beschreibung der Semantik von Daten kann auch die **Semantik von Operationen** über Ontologien beschrieben werden. Ein entsprechender Ansatz zur semantischen Beschreibung von Operationen wird von DeMeester [138] vorgestellt. Abbildung 6.8 stellt die hierzu verwendeten Konzepte und Beziehungen dar. Operationen (*Functions*) werden über ihre erwarteten Eingabeparameter (*Parameter*) und Rückgabewerte (*Output*) beschrieben. Zudem wird die Semantik der Operation über eine Beschreibung des gelösten Problems (*Problem*) spezifiziert und ggf. der verwendete Algorithmus (*Algorithm*) beschrieben. Zur Ausführung einer derart spezifizierten Operation müssen für die erwarteten Parameter konkrete Ausprägungen angegeben werden, die konform der Spezifikation sind (*Execution*).

Derart beschriebene Operationen können in **Funktions-Ontologien** zusammengefasst werden und somit eine semantische Beschreibung einer Gruppe von Operationen für bestimmte Anwendungsdomänen spezifiziert werden. Durch Referenzierung einer Operation in der Ontologie ist eindeutig festgelegt, welches Verhalten und welche Eingabeparameter und Rückgabewerte zur Ausführung erwartet werden.

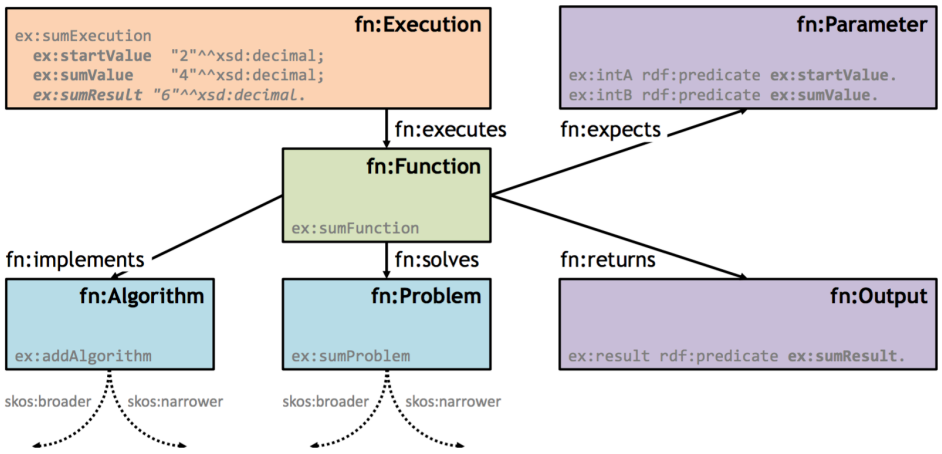


Abbildung 6.8. Konzepte der Funktions-Ontologie für Operationen (DeMeester et al.)

Der Ansatz von DeMeester et al. kann zur technologieneutralen Beschreibung des Verhaltens im Anwendungsmodell angewendet werden. Hierfür wird davon ausgegangen, dass die Operationen einer Anwendung in einer Menge von Funktions-Ontologien eindeutig spezifiziert wurden. Die Modellierung einer Operation erfolgt dann als Referenz auf die Funktions-Ontologie unter Angabe folgender Informationen:

- URI der Operation innerhalb der Funktions-Ontologie
- Eingabeparameter als Referenzen auf Daten des Anwendungsmodells
- Rückgabewerte als Referenzen auf Daten des Anwendungsmodells

Die Angabe der **URI der Operation** bestimmt, welche Operation gerufen werden soll. Sie referenziert eine Funktions-Ontologie, in welcher die Semantik der Operation und damit das erwartete Verhalten spezifiziert ist. In der Ontologie wird festgelegt, welche Parameter die Operation erwartet. Zur Laufzeit stammen die **Inhalte der Eingabeparameter aus dem Anwendungskontext**. Die Datenelemente können mittels der im vorangegangenen Abschnitt

beschrieben Pfadangabe referenziert werden. Analog kann für die **Ausgabeparameter** bestimmt werden, mit welchen Daten im Anwendungskontext sie korrespondieren.

In Abbildung 6.9 ist die Referenz zur Ermittlung eines Ortes über die Postleitzahl dargestellt. Die Operation wird hier mittels einer URI für die Operation *getCityForZIP* ① bestimmt. Die Operation entstammt einer exemplarischen Ontologie *Extended Type Operations Ontology* ②. Diese beschreibt die Semantik der Operation und die Ein-/Ausgabeparameter ③. Die Spezifikation des Aufrufs ① legt zudem fest, dass als Eingabeparameter das Datenelement *zipcode* der Adressdaten verwendet werden soll und dass das Ergebnis dem Datenelement *city* zugeordnet werden soll.

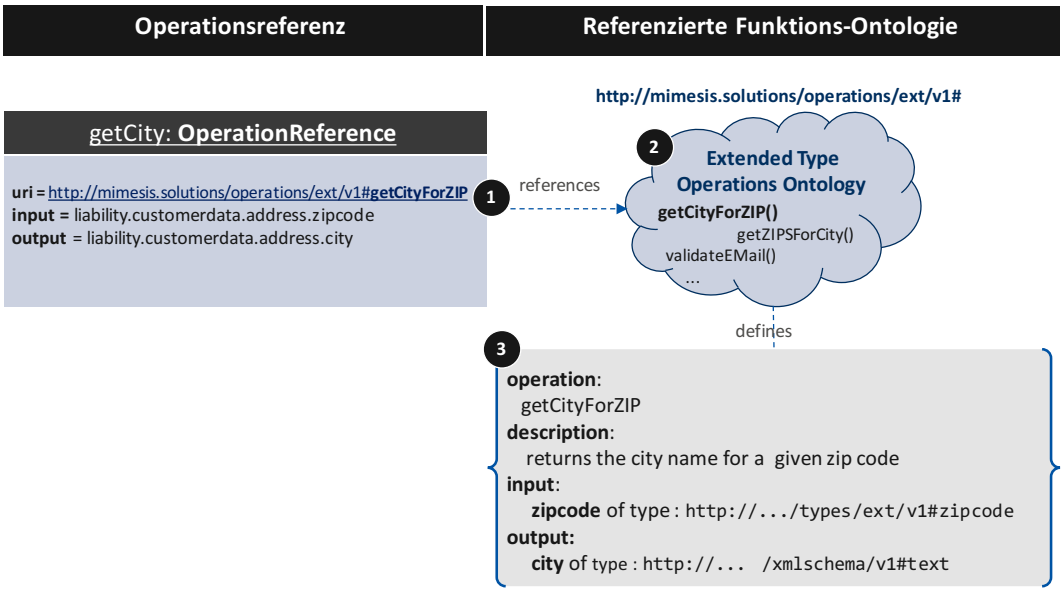


Abbildung 6.9. Referenzierung einer Operation aus einer Funktions-Ontologie

Der Vorteil des Ansatzes bei der Modellierung liegt in der **technologieneutralen Spezifikation von Operationsaufrufen**. Bei der Modellierung kann eindeutig festgelegt werden, welches Verhalten erwartet wird und welche Daten aus dem Anwendungsmodell beteiligt sind, ohne dabei auf eine konkrete Implementierung zu verweisen.

Durch die Externalisierung der Semantik von Operationen in Funktions-Ontologien werden zusätzlich die in Abschnitt 6.3.2 bei semantischen Annotationen genannten Vorteile erzielt. Auch für die Beschreibung von Operationen wird die **Offenheit und Erweiterbarkeit für beliebige Anwendungsdomänen** erreicht, ohne dadurch die generelle Nutzbarkeit des Anwendungsmodells einzuschränken. Funktions-Ontologien können für spezifische Anwendungsdomänen standardisiert und bereitgestellt werden, die im Anwendungsmodell frei kombiniert werden können.

6.3.4 Verhalten

Mit dem im vorangegangenen Abschnitt dargestellten Ansatz zur Modellierung von Operationen und Referenzen auf Datenelemente kann die Modellierung des Verhaltens einer Anwendung auf einfache Weise umgesetzt werden. In Abschnitt 5.5 wurden die Verhaltensarten identifiziert, die in dialogbasierte Benutzungsschnittstellen auftreten:

- Steuerung der Sicht-/Editierbarkeit von Informationen (IR.10)
- Validierung eingegebener Daten (IR.11)
- Reaktionen auf Änderung im Anwendungskontext (IR.12)
- Elementinitiierte Aktionen (IR.13)
- Nutzer- und systeminitiierte Aktionen (IR.14)

Tabelle 5.3 stellt die Informationen dar, die zur Beschreibung des jeweiligen Verhaltens benötigt werden. Hier ist aufgeführt, welchem Element ein Verhalten zugeordnet ist, welche Auslöser das Verhalten anstoßen, welche Informationen als Eingabe benötigt werden und welche Wirkung das Verhalten hat. In Abbildung 6.10 ist die Modellierung dieser Eigenschaften im Metamodell dargestellt. Verhalten wird als Operation (*Operation*) modelliert, welche *DescriptionElements* zugewiesen wird. Anhand der in Tabelle 5.3 aufgeführten Informationen können vier unterschiedliche Arten von Operationen abgeleitet werden, deren Inhalte in Abbildung 6.10 als Klassen dargestellt sind.

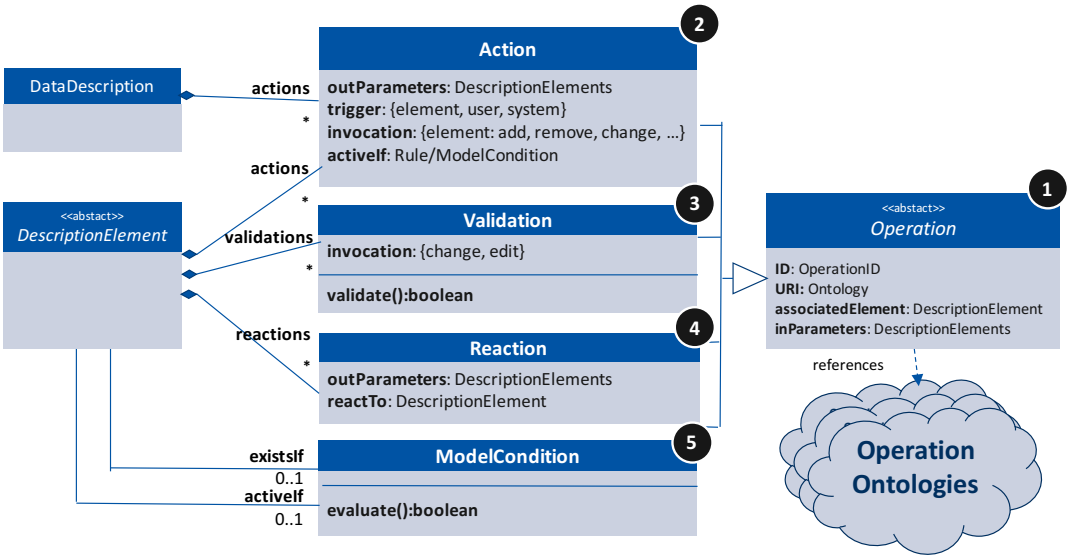


Abbildung 6.10. Modellierung verhaltensrelevanter Informationen im Metamodell

Die Klasse ***Operation*** ① fasst als abstrakte Basisklasse alle Attribute zusammen, die alle Operationen gemeinsam besitzen. Die Attribute beschreiben die auszuführende Operation (*ID*, *URI*) und die Daten im Anwendungskontext, die als Eingabeparameter (*inParameters*) verwendet werden sollen. Zudem ist angegeben, welchem Element im Strukturbaum die Operation zugeordnet wird (*associatedElement*). Die Angabe der Referenzen auf Operationen und Daten erfolgt wie in Abschnitt 6.3.3 beschrieben. Datenelemente werden über ihren Pfad im Strukturbaum, Operationen über eine Referenz auf eine Ontologie beschrieben.

Aktionen (**Actions** ②) beschreiben Operationen, die Auswirkungen auf den Anwendungskontext haben. Hierzu werden in *outParamaters* die Datenelemente des Anwendungsmodells angegeben, die mit Rückgabewerten der Operation korrespondieren. Über das Attribut *trigger* können die Auslöser der Aktion angegeben werden. Dies kann das assoziierte Element, ein Nutzer- oder ein Systemereignis sein (vgl. Tabelle 5.3). Über das Attribut *invocation* kann zudem genauer spezifiziert werden, welches Ereignis die Operation auslöst. Für ein Element kann z.B. angegeben werden, ob die Operation beim Aktivieren/Deaktivieren oder Editieren/Löschen des Feldes oder Inhalts ausgelöst werden soll.

Reaktionen (**Reactions** ④) beschreiben Operationen, die auf das assoziierte Element wirken und werden durch Änderungen am Anwendungskontext ausgelöst. Hierzu wird das Element im Anwendungskontext referenziert, auf dessen Änderung reagiert werden soll (*reactTo*).

Validierungen (**Validations** ③) beschreiben Operationen zur Validierung eines Elements. Hier wird angegeben, bei welchem Ereignis die Operation ausgelöst wird (z.B. während oder nach Abschluss der Eingabe). Die Operation liefert einen Wahrheitswert (*boolean*), der angibt, ob die Validierung erfolgreich verlief.

Modellbedingungen (**ModelConditions** ⑤) beschreiben Prüfungen, die auf Daten im Anwendungskontext durchgeführt werden. Analog den Validierungen liefern diese einen Wahrheitswert (*boolean*) als Ergebnis, der anzeigt, ob die Bedingung erfüllt ist.

Mittels dieser Operationsklassen kann das Verhalten modelliert werden. In Abbildung 6.11 sind Beispiele in Form eines UML-Instanzdiagramms angegeben.

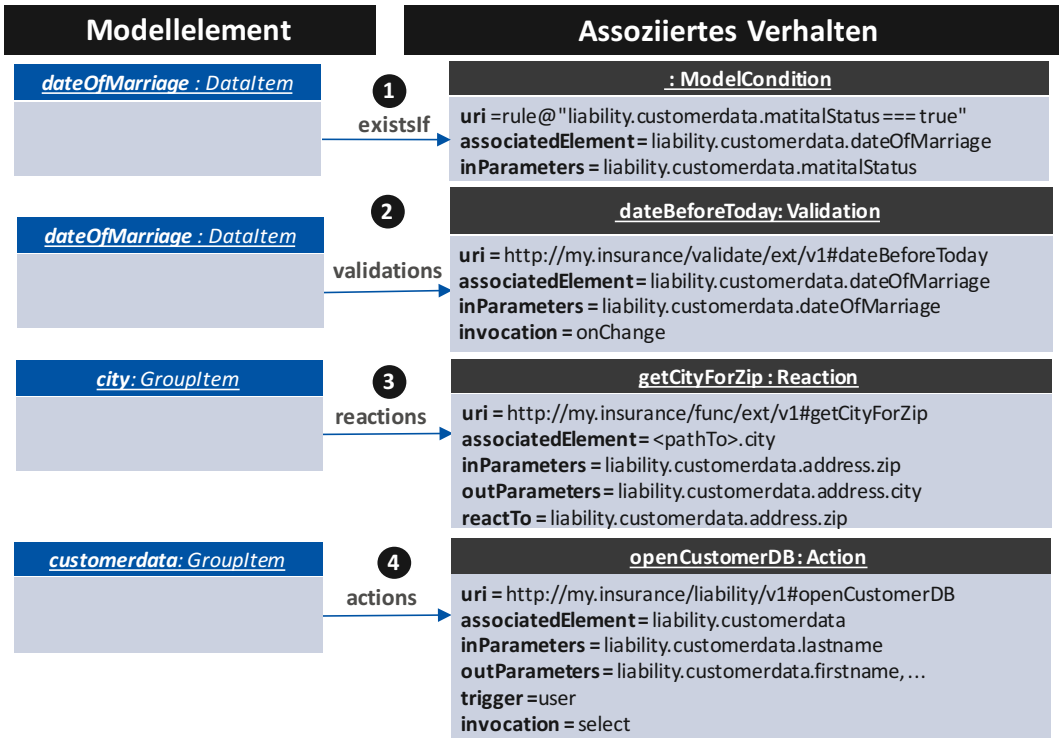


Abbildung 6.11. Beispiele für die Modellierung des Verhaltens

Steuerung der Sicht- und Editierbarkeit

Die Modellierung erfolgt über eine Modellbedingung (*ModelCondition*). Jedem *DescriptionElement* kann eine Modellbedingung für die Existenz bzw. die Editierbarkeit zugewiesen werden (Abbildung 6.10, *existsIf* bzw. *activeIf* -Relation). Diese bestimmt über die Eingabeparameter (*inParameters*), welche Daten des Anwendungsmodells beteiligt und damit auch Auslöser der Sichtbarkeitsbestimmung sind. Die Operation liefert einen Wahrheitswert zurück, welcher die Sicht-/Editierbarkeit angibt und damit auf den Zustand des Elements selbst wirkt. Diese Modellierung enthält alle in Tabelle 5.3, (IR.10) identifizierten Informationen.

Abbildung 6.11, ① zeigt die Modellierung einer Sichtbarkeitsbedingung für das Datenelement *dateOfMarriage*, dessen Sichtbarkeit durch eine Regel in Abhängigkeit des Familienstandes (*maritalStatus*) bestimmt wird.

Validierung eingegebener Daten

Die Modellierung erfolgt über eine Liste von Validierungsoperationen (*Validation*), die einem *DescriptionElement* zugewiesen werden (Abbildung 6.10, *validations*-Relation). Über die Angabe eines Auslösers (*invocation*) wird festgelegt, welches Ereignis die Validierung auslöst. Die Eingabeparameter (*inParameters*) beschreiben, welche Kontextinformationen beteiligt sind. Die ausgeführte Operation liefert als Ergebnis die Validität des Elements zurück und wirkt auf den Zustand des Elements selbst (*gültig, ungültig*). Diese Modellierung enthält alle in Tabelle 5.3, (IR.11) identifizierten Informationen.

Abbildung 6.11, ② zeigt die Modellierung der Validierung für das Datenelement *dateOfMarriage*, welches über die Operation *dateBeforeToday* validiert wird. Diese wird aufgerufen, sobald die Eingabe abgeschlossen ist (*invocation=onChange*).

Elementreaktionen bei Änderung des Anwendungskontexts

Die Modellierung erfolgt über Reaktionsoperationen (*Reaction*), die einem *DescriptionElement* zugewiesen werden (Abbildung 6.10, *reactions*-Relation). Diese beschreiben, auf Änderung welchen Datums im Anwendungskontext regiert wird (*reactTo*). Die Operation wirkt auf den Zustand des assoziierten Elements. Diese Modellierung enthält alle in Tabelle 5.3, (IR.12) identifizierten Informationen.

Abbildung 6.11 ③ zeigt die Modellierung der Reaktion für das Datenelement *city*, welches auf Änderungen des Datenelements *zip* reagiert (_reactTo = *zip*). Die Reaktion besteht im Aufruf der Operation *getCityForZip*, welche den Inhalt von *city* anpasst.

Element-, nutzer- und systeminitiierte Aktionen

Element-, nutzer- und systeminitiierte Aktionen können über den selben Satz an Informationen beschrieben werden. Gemäß Tabelle 5.3 bestehen die Unterschiede lediglich in der Art des Auslösers (Element, Nutzer oder System) und dem auslöserspezifischen Ereignis.

Die Modellierung erfolgt daher einheitlich über Aktionsoperationen (*Action*), die einem *DescriptionElement* zugewiesen werden (Abbildung 6.10, *actions*-Relation). Sie beschreiben, durch welchen Auslöser (*trigger*) und welches Ereignis (*invocation*) die Operation angestoßen wird und auf welche Daten im Anwendungskontext sie wirkt (*outParamaters*). Diese Modellierung enthält alle in Tabelle 5.3, (IR.13) und (IR.14) identifizierten Informationen.

Abbildung 6.11 ④ zeigt die Modellierung einer nutzerinitiierten Aktion (*trigger = user*) für das Gruppenelement *customerdata*, welche durch Auswahl z.B. aus einem Menü (*invocation = select*) einen Kundenauswahl-Dialog öffnet (*customerSelection*). Die Operation liefert Kundendaten, die in den Anwendungskontext übertragen werden (*outParameters=...*).

6.3.5 Resultierendes Gesamtmodell

Abbildung 6.12 fasst die Ergebnisse der vorangegangenen Abschnitte als Gesamtmodell in einem UML-Diagramm zusammen. Die Elemente des Metamodells sind nach den Anforderungsbereichen gruppiert dargestellt, zu deren Umsetzung sie beitragen.

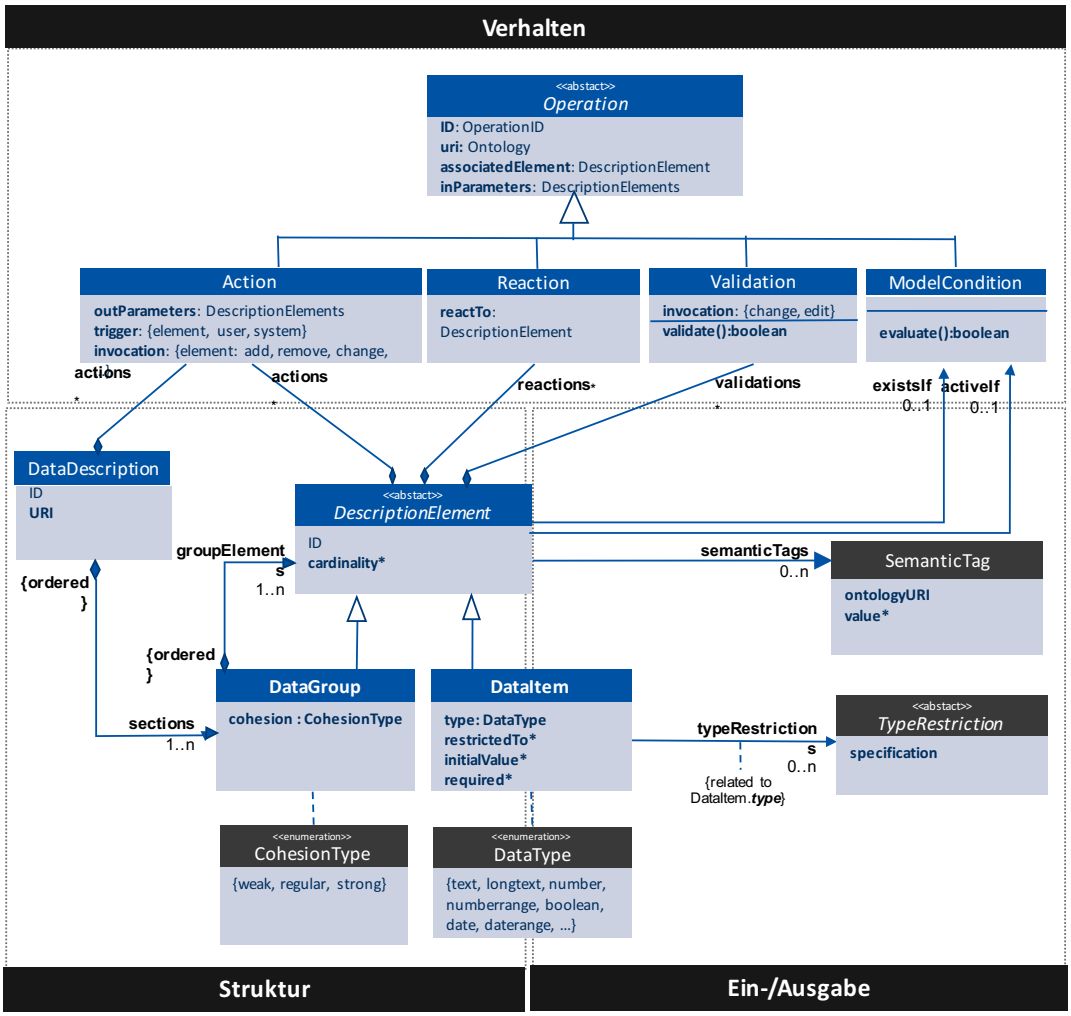


Abbildung 6.12. Metamodell zur Beschreibung dialogbasierter Anwendungen

Die Modellierung von **Struktur und Aufbau** der Benutzungsschnittstelle erfolgt in einer Anwendungsbeschreibung (*DataDescription*) über eine Folge von Beschreibungselementen (*DescriptionElements*), die in Gruppen (*DataGroups*) zusammengefasst werden. Gruppen können wiederum eine Folge von Gruppen bzw. Datenelemente (*DataItems*) enthalten. Zudem sind Informationen zum Grad der semantischen Kohäsion in einer Gruppe enthaltener Daten enthalten. Hierdurch wird die hierarchische Struktur der Daten, deren Kohäsion sowie die temporale Abfolge der Fragen beschrieben, welche den Aufbau der Benutzungsschnittstelle bei der automatischen Generierung bestimmt. **Mit dieser Modellierung sind die Informationen zur Umsetzung der Anforderungen (IR.1), (IR.2) und (IR.3) aus Tabelle 5.1 im Modell enthalten.**

Die Beschreibung der **Ein- und Ausgabe** erfolgt durch Attribuierung der Beschreibungselemente. Es können grundlegende Typinformationen für Gruppen und Datenelemente spezifi-

ziert und Datenelemente über Restriktionen (*TypeRestriction*) näher bestimmt werden. Die Beschreibung kann über die Angabe von semantischen Annotationen (*SemanticTags*) domänenspezifisch verfeinert werden. Bei der automatischen Generierung kann hieraus abgeleitet werden, welche Interaktionselemente zur Ein- und Ausgabe verwendet werden können. **Mit dieser Modellierung sind die Informationen zur Umsetzung der Anforderungen (IR.5), (IR.6), (IR.7) und (IR.8) aus Tabelle 5.2 im Modell enthalten.**

Die Modellierung des **Verhaltens** von Modellelementen erfolgt über die Angabe von Operationen für die Verhaltensarten. Für Darstellungselemente können Bedingungen für die Existenz und Editierbarkeit angegeben (*ModelCondition*) sowie Operationen für Validierung (*Validations*), Reaktionen (*Reactions*) oder Aktionen (*Actions*) über auslösende Ereignisse und benötigte Daten aus dem Anwendungskontext spezifiziert werden. **Mit dieser Modellierung sind die Informationen zur Umsetzung der Anforderungen (IR.10)-(IR.14) aus Tabelle 5.3 im Modell enthalten.**

6.4 Metamodell zur Beschreibung inhaltlicher Varianten

Zur Entwicklung des Metamodells für Varianten wurde anhand der Anforderungen zu inhaltlichen und technischen Varianten identifiziert, welche Elemente des Anwendungsmodells variieren und eine Methodik zur Beschreibung dieser Unterschiede entwickelt.

In den Tabellen 5.1, 5.2 und 5.3 (Kapitel 5) sind die Anforderungen an **technische und inhaltliche Varianten** zusammengefasst (vgl. IR.3, IR.4, IR.8, IR.9, IR.15, IR.16). Die Erzeugung technischer Varianten ist dabei unabhängig von der Fachlichkeit der Anwendung und kann weitgehend automatisch erfolgen. Die Unterschiede bei inhaltlichen Varianten hingegen lassen sich nicht automatisch ableiten und müssen Teil einer Variantenbeschreibung sein. Hierzu sind Informationen zur

- **Auswahl relevanter Elemente** für eine Variante (IR.4)
- **Modifikation von Eigenschaften** einzelner Elemente (IR.9)
- **Modifikation des Verhaltens** einzelner Elemente (IR.15) und (IR.16)

erforderlich. Diese Informationen können in einem Variantenmodell beschrieben werden, das sowohl die Auswahl variantenspezifischer Elemente (IR.4) als auch die Modifikation von Elementen des Anwendungsmodells definiert (IR.9, IR.15, IR.16). Das zugrundeliegende Metamodell für eine Variantenbeschreibung ist in Abbildung 6.13 als UML-Diagramm dargestellt.

Eine Variantenbeschreibung (*VariantDescription* ①) bezieht sich auf ein spezifisches Anwendungsmodell (*datamodelURI*), dessen Elemente in der Variante verändert werden.

Zur Auswahl für die Variante relevanter Modellelemente werden Regeln angegeben (*ElementRule* ②), die aus dem zugrundegelegten Anwendungsmodell einzelne Elemente ein- bzw. ausschließen. Für jedes Element des Anwendungsmodells können zudem Modifikatoren (*ElementModifier* ③) angegeben werden, die einzelne Eigenschaften des Elements anpassen (s. Abschnitte 6.4.1 und 6.4.2).

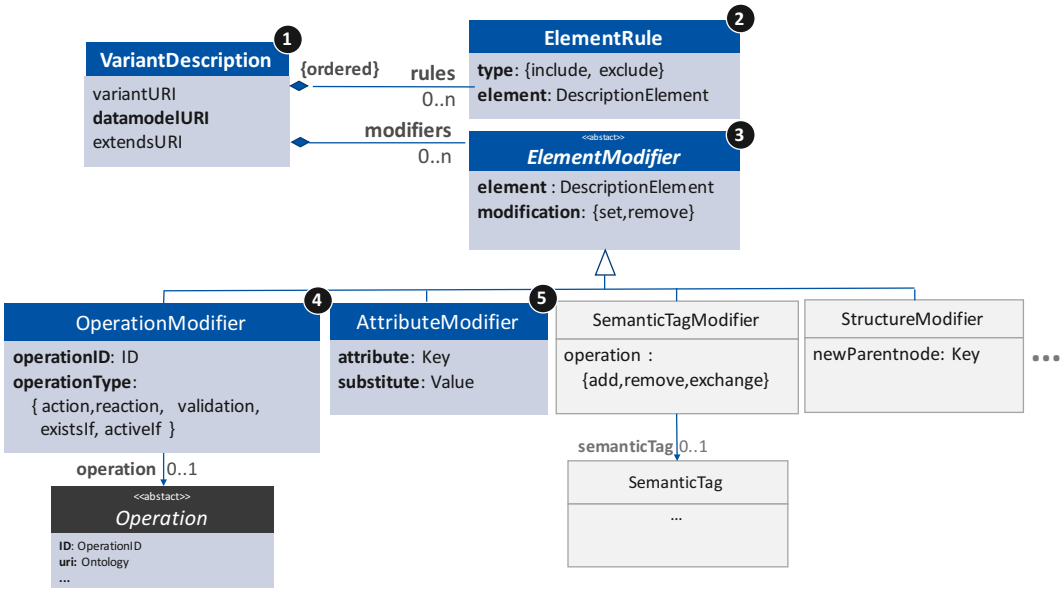


Abbildung 6.13. Metamodell zur Beschreibung von Anwendungsvarianten

6.4.1 Auswahl relevanter Elemente

Die Auswahl von Elementen kann als Operation auf dem Strukturbaum des Anwendungsmodells (Abschnitt 6.3.1) aufgefasst werden. Dieser enthält die Gesamtmenge der Gruppen und Datenelemente der Anwendung, die über eine Pfadangabe referenziert werden können. Die Auswahl relevanter Elemente für eine Variante erfolgt durch explizite *Inklusion bzw. Exklusion von Knoten des Strukturbaums*. Hierzu werden im Gesamtbaum alle Knoten markiert, die in der Variante enthalten bzw. ausgeschlossen sein sollen.

Zur Modellierung der In- bzw. Exklusion werden Elementregeln (*ElementRules*) verwendet, welche als *geordnete Liste* in der Variantenbeschreibung angegeben sind (*rules*, Abbildung 6.13 ②). Eine Elementregel referenziert einen Knoten anhand seines Pfades im Strukturbaum und gibt über den Regeltyp (*type*) an, ob der Knoten *in- oder exkludiert* werden soll. Die Ordnung der Liste bestimmt dabei, in welcher Reihenfolge die Regeln angewendet werden.

Abbildung 6.14 (links) zeigt exemplarisch den Ausschnitt des Strukturbaums für Kundendaten, für welchen eine vereinfachte Variante für Endkunden erstellt werden soll. Hierbei soll auf die Geburtsdaten verzichtet ① und die Kontaktinformation auf die E-Mail-Adresse reduziert werden ② ③. In der Abbildung (rechts) ist die Variantenbeschreibung in Form eines UML-Instanzdiagramms dargestellt. Hier wird eine Exklusionsregel angegeben, welche die Geburtsdaten ausschließt ①. Über weitere Exklusionsregeln werden zuerst die Kontaktdaten ausgeschlossen ② und anschließend die E-Mail-Adresse wieder hinzugenommen ③.

6.4.2 Modifikation von Elementeigenschaften und Verhalten

Elementeigenschaften sind als direkte Attribute in Strukturbaumknoten vermerkt, Operationen werden über Operations-Objekte beschrieben, die mit dem Knoten assoziiert sind. Die Anpassung der Elementeigenschaften oder des Verhaltens kann daher als Modifikation der Attribute des entsprechenden Knotens im Strukturbaum aufgefasst werden.

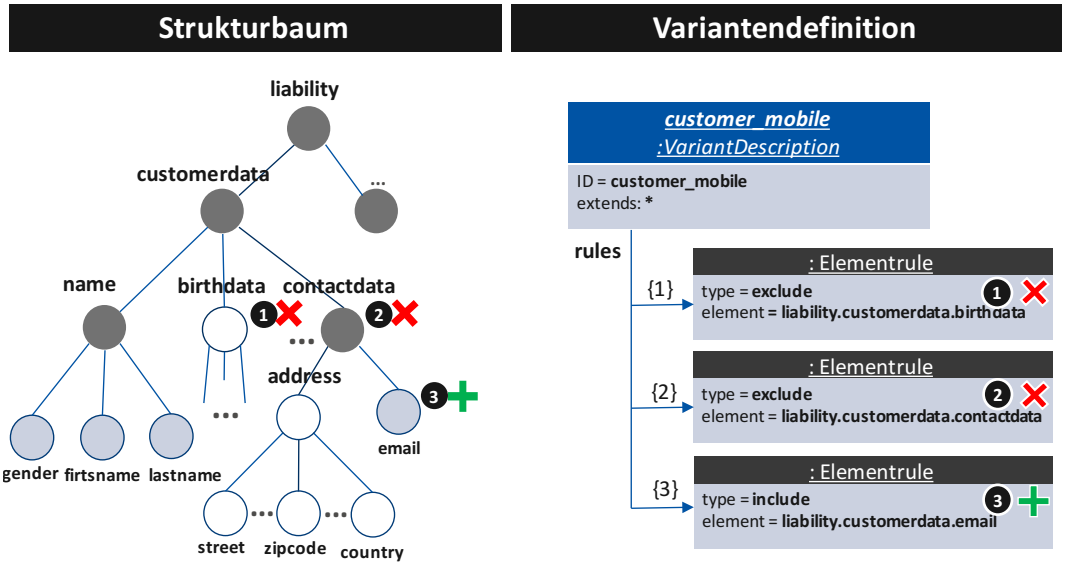


Abbildung 6.14. In- und Exklusion von Teilbäumen bzw. Elementen in Varianten

Zur Modellierung von Attributänderungen eines Strukturbaumknotens werden in der Variantenbeschreibung in Abbildung 6.13 **Modifikatoren** angegeben (*ElementModifier* ③). Ein Modifikator bestimmt durch Angabe des Pfades das Element, auf welchem die Modifikation ausgeführt werden soll (*element*). Zudem wird angegeben, ob die Eigenschaft gesetzt oder entfernt werden soll (*modification*).

Zur **Modifikation der Elementeigenschaften** wird ein **Attributmodifikator** verwendet (Abbildung 6.13 ⑤ *AttributeModifier*). Dieser bestimmt das zu modifizierende Knotenattribut (*attribute*) und den zu ersetzenden Wert (*substitute*). Die **Modifikation des Verhaltens** erfolgt mittels eines **Operationsmodifikators** (Abbildung 6.13, *OperationModifier* ④). Dieser spezifiziert die zu modifizierende Verhaltensart (*operationType*, z.B. *action*, *reaction* oder *existsIf*) und die zu setzende Operation (*operationID*). Es wird eine Operationsbeschreibung angegeben (*operation*), die in der Variante verwendet werden soll.

Abbildung 6.15 (links) zeigt exemplarisch den Ausschnitt des Strukturbaums für Kundendaten. Es soll eine Variante für verheiratete Personen erstellt werden, in welcher die Partnerdaten im Fragefluss erscheinen und keinesfalls auf eine Unterseite ausgelagert werden dürfen ①. Der Familienstand soll fest auf *verheiratet* gesetzt und nicht mehr zur Auswahl angeboten werden ②. Da die Partnerdaten durch diese Änderung immer angezeigt werden, soll die Existenzbedingung (*existsIf*) des Knotens entfernt werden ③. Zudem soll eine Operation zur Vorbelegung der Daten gerufen werden. Diese wird durch das System ausgelöst, wenn der Benutzer sich anmeldet ④.

In der Abbildung (rechts) ist die Variantenbeschreibung in Form eines UML-Instanzdiagramms dargestellt. Hier wird zur Anpassung der Partnerdaten eine Attributmodifikation angegeben, welche deren Kohäsion auf den Wert *default* setzt ①. Über weitere Attributmodifikationen ② wird zuerst der Familienstand vorbelegt (*initialValue* = *married*) und anschließend das Element dauerhaft ausgeblendet (*hidden* = *true*). Im nächsten Schritt entfernt eine Operationsmodifikation ③ die Existenzbedingung der Partnerdaten. Eine weitere Operationsmodifikation ④ setzt am Knoten für Kundendaten eine Aktion

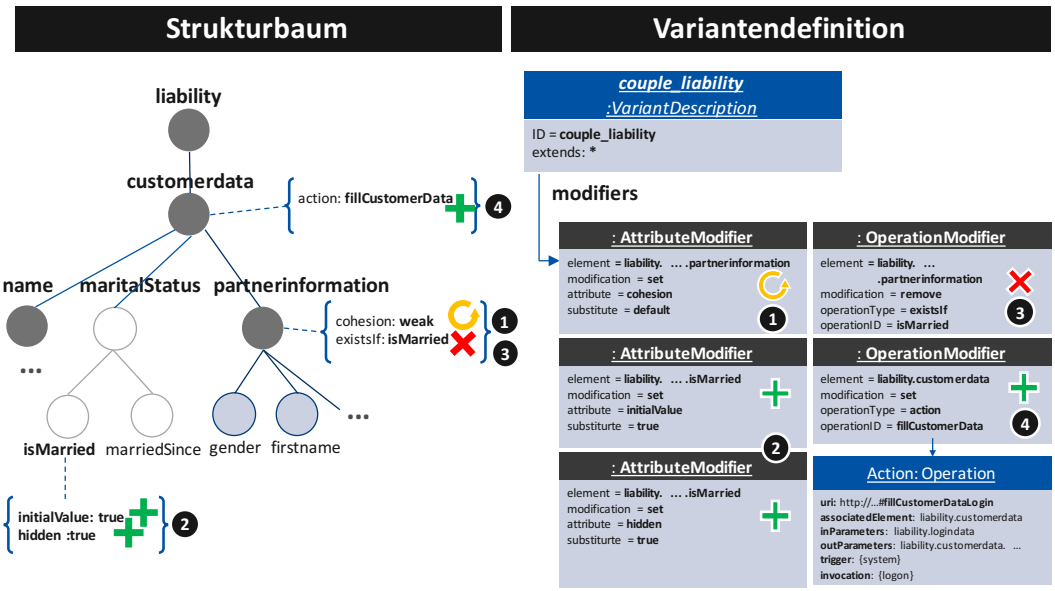


Abbildung 6.15. Modifikation von Elementeigenschaften und Verhalten

(*fillCustomerData*), welche durch das System bei einer erfolgten Anmeldung angestoßen wird.

6.5 Repräsentation als *Domain Specific Language* (mimesis.DSL)

Die Metamodelle in Abschnitt 6.3 und 6.4 bilden die Grundlage zur Erstellung von Sprachen, mit denen Benutzungsschnittstellen und deren Varianten beschrieben werden können. Zur Modellierung von Anwendungen wird eine Umsetzung des Metamodells in eine Notation benötigt, welche konkrete Instanzen beschreibt. Parallel zur Umsetzung der Metamodelle wurde als Notation eine *Domain Specific Language* (DSL) entwickelt, die sich auf zwei Artefakte beschränkt:

- eine **Anwendungsbeschreibung** und
- eine **Variantenbeschreibung**.

Die **Anwendungsbeschreibung** beinhaltet das Anwendungsmodell und stellt eine textuelle Repräsentation des in Abschnitt 6.3 vorgestellten Metamodells dar. Die **Variantenbeschreibung** beschreibt inhaltliche Varianten des Anwendungsmodells und beinhaltet die in Abschnitt 6.4 dargestellten Informationen.

Im Folgenden wird die Notation exemplarisch dargestellt. Eine detaillierte Beschreibung der DSLs erfolgt in Anhang A.3. Dort ist deren Aufbau beschrieben und die Grammatik der Notationen angegeben.

6.5.1 Notation für Anwendungsbeschreibungen

Die DSL der Anwendungsbeschreibung modelliert den Strukturbaum der Anwendung über Gruppen und Datenelemente und ordnet diesen Attribute zu. Listing 6.1 zeigt den Ausschnitt einer Anwendungsbeschreibung am Beispiel der Kundendaten in einer Antragstrecke.

```
data_description : "liability"
❶uri="http://mimesis.solutions/mimesis_corp/liability/v2#"
  referencedOntologies=[
      ext:http://mimesis.solutions/types/ext/v1#,
      userdb: http://mimesis.solutions/agent/operations/dataaccess/v3#,
      zipcity: http://mimesis.solutions/types/ext/operations/v3#]
{ ...
❷ group : "customerdata" {
      actions="user:select:userdb.chooseCustomer($liability.customerdata)"}
❸  {
      group : "fullname"
      {
❹       gender : { restrictedTo="male|female" sematicTags="ext:gender" }
        firstname : {required}
        lastname  : {required}
      }
      group : "birthdata"
      {
        dateofbirth : { type="date" initialValue="1970-01-01T00:00:00.000" required}
        placeofbirth : {}
      }
      group : "maritalinformation"
      {
        status :{restrictedTo="married|notmarried|divorced" initialValue="notmarried"}
        marriedsince : {type="date"
           existsIf="(liability.customerdata.maritalinformation.status == 'married')" }
      }
❺     group : "partnerinformation"  {
          existsIf="(liability.customerdata.maritalinformation.status == 'married')"
          cohesion="weak" }
      { ... }
      group : "children"
      {
           numberOfChildren : {
           type="number" min="0" max="10"
❻          validations="validation.checkChildrenAmount(
                        liability.customerdata.children.numberOfChildren)"}
        ...
      }
      group : "address"  { cohesion="weak"}
      {
        street : {}
        building_no : {semanticTags="ext:buildingno"}
        zip     : {semanticTags="ext:zipcode"}
❼       city    : {reactions="liability.customerdata.address.zip:zipCity.prefillCity(...
    ↪ liability.customerdata.address.zip,$liability.customerdata.address.city)"}
        country : {semanticTags="ext:country" restrictedTo="zipCity.getCountries()"}
        eMail   : {semanticTags="ext:email" required}
        phone   : {semanticTags="ext:phone"}
      }
    } ... }
```

Listing 6.1: Beispiel einer Anwendungsbeschreibung in mimesis.DSL-Notation

Die Datenbeschreibung beginnt mit der Angabe einer eindeutigen Identifizierung des Anwendungsmodells über eine eindeutige *URI* und der Angabe im weiteren Verlauf referenzierter Ontologien zur Angabe der Semantik von Daten bzw. Operationen ①.

Die Gruppierung und innere Kohäsion (vgl. Abbildung 6.12, *Struktur*) wird durch die Zusammenfassung semantisch zusammenhängender Elemente in *group*-Bereichen beschrieben ②. Jede Gruppe besitzt einen Inhalts-Block ③, in welchem Elemente und weitere Gruppendeklarationen zusammengefasst werden ④. Hierdurch wird die hierarchische Struktur des Metamodells abgebildet. Die Sequenz der Elemente ist über die Reihenfolge innerhalb der Gruppe beschrieben.

Gruppen und Datenelemente werden in Attribut-Blöcken über Attribut-Wert-Paare näher spezifiziert. Die möglichen Attribute entsprechen den in Anhang A.2.1, Tabelle A.3-A.5 aufgeführten Angaben für Gruppen und Elemente. Listing 6.1 ⑤ zeigt exemplarisch Gruppen-Attribute für Partnerinformationen (*partnerinformation*). Der Gruppe wird eine *schwache äußere Kohesion* zugewiesen (*cohesion="weak"*) und sie besitzt eine Existenzbedingung. Listing 6.1 ④ zeigt exemplarisch Typinformationen und Restriktionen für ein Datenelement (*gender*). Hier existiert einerseits eine Werteauswahl-Restriktion (*restrictedTo*) und die Angabe von semantischen Annotationen zur näheren Beschreibung des Typs (*semanticTags*).

Das Verhalten der Benutzungsschnittstelle wird ebenfalls über Attribute bestimmt ②⑤⑥⑦. Für Existenz-/Sichtbarkeitsbedingungen, Validierungen, Aktionen und Reaktionen werden hierfür Attribute gemäß Anhang A.2.1, Tabelle A.4 verwendet. Die Operationen und Zusatzinformationen (z.B. Name und URI der Operation, Ein-/Ausgabeparameter und auslösende Ereignisse) werden in textueller Form angegeben. Hierbei wird eine kompakte Schreibweise verwendet, die alle benötigten Informationen für den jeweiligen Operationstyp enthält (vgl. Anhang A.3.1). Zur eindeutigen Referenzierung von Modellelementen im Anwendungskontext wird die in Abschnitt 6.3.3 dargestellte Pfadnotation verwendet.

Eine erweitertes Bespiel für eine Anwendungsbeschreibung ist in Anhang A.8.1 angegeben. Eine detaillierte Beschreibung der DSL und ihrer Syntax befindet sich in Anhang A.3.1. Zur interaktiven Untersuchung einer lauffähigen Variante werden in Anhang A.7 Verweise angegeben.

6.5.2 Notation für Variantenbeschreibungen

Für die Beschreibung inhaltlicher Varianten wird analog der DSL zur Anwendungsbeschreibung eine textuelle Repräsentation des Metamodells aus Abschnitt 6.4 verwendet. Listing 6.2 zeigt einen Ausschnitt der Variantenbeschreibung für Endkunden, die für die Nutzung in einem Portal vorgesehen ist (*customer_internet*).

Die Variantenbeschreibung ① beginnt mit der Identifizierung der Variante über eine eindeutige URI und der Angabe, auf welches Anwendungsmodell sie anwendbar ist (*datamodelURI*). Zudem werden Ontologien angegeben, die im Verlauf der Variantenbeschreibung genutzt werden (*referencedOntologies*).

Inhaltliche Varianten werden durch eine Folge von In- und Exklusionsregeln spezifiziert ②, die auf dem in der Anwendungsbeschreibung enthaltenen Strukturbaum basieren. Die Identifikation des betroffenen Knotens im Strukturbaum erfolgt mittels der in Abschnitt 6.3.3 eingeführten Pfadnotation. Für den referenzierten Knoten wird angegeben, ob er in der Variante in- oder exkludiert werden soll.

In der Variante erforderliche Elementmodifikationen werden durch Angabe von Modifikations-Beschreibungen (*modify*) vorgenommen. Listing 6.2 ③ zeigt dies beispielhaft

durch setzen einer Reaktion. Die Bezeichnungen der Beschreibung entsprechen den im Metamodell beschriebenen Angaben zu Operationsmodifikationen für Reaktionen.

```
variant_description : {
❶ variant : "customer_internet" {
    uri : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#customer_internet"
    datamodelURI : "http://mimesis.solutions/mimesis_corp/liability/v2"
    referencedOntologies : [libzip : mimesis.solutions/mimesis_corp/functions/...
        ↪ international/v1#]

❷ // inhaltliche Modifikation
    exclude : "liability.customerdata.birthdata.placeofbirth"
    exclude : "liability.customerdata.maritalinformation.marriedsince"
    exclude : "liability.customerdata.partnerinformation"
    ...
    exclude : "liability.contractdata.previousdamages.damagedescription"
    exclude : "liability.contractdata.productconfiguration.component_oiltank"
    include : "liability.contractdata.productconfiguration.component_oiltank.oiltankplus"
    exclude : "liability.contractdata.productconfiguration.component_landlord"
    ...

❸ // Operationsmodifikation
    modify:{
      type : "operation"
      element : "liability.customerdata.address.city"
      modification : "set"
      operationType : "reaction"
      operationID : "prefillCity"
      reaction_reactto : "liability.customerdata.address.zip"
      reaction_operation : "libzip:prefillInternationalCity"
      reaction_parameters : "liability.customerdata.address.zip, $liability.customerdata....
          ↪ address.city"}

❹ // Attributmodifikation
    modify:{
      type : "attribute"
      element : "liability.contractdata.kindofliability"
      attribute : "hidden"
      newvalue : "true" }
    modify:{
      type : "attribute"
      element : "liability.contractdata.kindofliability"
      attribute : "initialValue"
      newvalue : "family" }
  }
}
```

Listing 6.2: Beispiel einer Variantenbeschreibung in mimesisDSL-Notation

Listing 6.2 ④ zeigt die Modifikation von Elementeigenschaften. Das Element *liability.contractdata.kindofliability* wird hier mit dem Wert vorbelegt (Modifikation von *initialValue*) und das Interaktionselement unsichtbar gesetzt (Modifikation des Attributs *hidden*).

Ein erweitertes Beispiel für Variantenbeschreibungen ist in Anhang A.8.2 angegeben. Eine Beschreibung der DSL und ihrer Syntax befindet sich in Anhang A.3.2. Zu interaktiven Untersuchung lauffähiger Varianten siehe Verweise in Anhang A.7.

6.5.3 Alternative Repräsentationen

Die dargestellte Umsetzung als DSL fokussiert auf den menschlichen Nutzer. Sie erlaubt eine kompakte Erstellung und Lesbarkeit mit einfachen Mitteln (ein Texteditor genügt). Ein Nachteil von DSLs ist, dass sie ein proprietäres Format darstellen. Dies schränkt die Wiederverwendbarkeit in unterschiedlichen Kontexten ein. Insbesondere wird die maschinelle Verarbeitung durch das proprietäre Format erschwert, da nicht auf standardisierte Technologien zur Verarbeitung der Informationen zurückgegriffen werden kann.

Das vorgestellte Metamodell ist unabhängig von einer spezifischen Notation, sodass es auf weitere gängige bzw. standardisierte Sprachen übertragen werden kann. Die Voraussetzung ist lediglich, dass mit der Sprache ein *attribuierter Strukturbaum* beschrieben werden kann. Geeignete Sprachen zur Beschreibung sind z.B. **grafische Notationen** (z.B. Unified Modeling Language [157]) und **textuelle Notationen** (z.B. Extensible Markup Language (XML/XML-Schema) [61], JavaScript Object Notation JSON). Zudem können **Sprachen zur Wissensrepräsentation** verwendet werden (RDF, OWL, Turtle, JSON-LD).

Im Verlauf der Arbeit wurden neben der dargestellten Notationen weitere Darstellungsoptionen untersucht und umgesetzt. So existieren beispielsweise alternative Repräsentationen des Anwendungsmodells in JSON-Notation und als XML-Schema-Beschreibung (Moosbauer [147]). Zur Nutzung der Beschreibung als *Shared UIs* wurde zudem eine Umsetzung des Metamodells in Form von Ontologien in RDF/OWL-Notation erstellt (Hitz et al. [99, 100]). Diese Beschreibung eignet sich insbesondere zur Erstellung wiederverwendbarer Modelle und dient in Kapitel 9 als Grundlage.

6.6 Diskussion der Ergebnisse

Ziel des Kapitels war die Darstellung eines Modellierungsansatzes für dialogbasierte Benutzungsschnittstellen und deren inhaltlicher und technischer Varianten. Die grundsätzlichen Anforderungen an die Modellierung bestanden in einer geringen Komplexität (geringe Artefaktzahl und Abhängigkeiten) sowie deren Technologieneutralität, um die Anwendbarkeit auf beliebige Interaktions- und Technologiekontexte zu erreichen. Die inhaltlichen Anforderungen wurden in Abschnitt 3.1, Tabelle 3.1 unter **(A.1)** mit der Vollständigkeit des Modells **(A1.1)**, der Eignung zur Erzeugung technischer Varianten **(A1.2)** und der Beschreibung inhaltlicher Varianten **(A1.3)** subsummiert.

Hierzu wurde ein Metamodell präsentiert welches dialogbasierte Benutzungsschnittstellen datenzentriert als Modell der verarbeiteten Daten über deren Struktur und Aufbau, typisierter Ein-/Ausgabe und Verhalten beschreibt. Zur Umsetzung der Vollständigkeit (A1.1) wurden alle Informationen in das Metamodell überführt, die in Abschnitt 5, Tabellen 5.1-5.4 als notwendig zur Generierung identifiziert wurden. Struktur und Aufbau der von der Anwendung verarbeiteten Daten werden in einem Anwendungsmodell als attribuierter Strukturbaum beschrieben, dessen Knoten um Informationen zum Typ und Semantik sowie Verhalten angereichert wurden (vgl. Abschnitte 6.3.1, 6.3.2, 6.3.4).

Zur Erzeugung nicht-trivialer technischer Varianten (A1.2), wurde ein Ansatz zur semantischen Beschreibung von Daten und Datengruppen über **semantische Annotationen** vorgestellt. Die Annotationen können von Generatoren zur Erstellung optimierter bzw. domänenspezifischer Interaktionselemente ausgewertet werden (vgl. Abschnitt 6.3.2). Zur Beschreibung des Verhaltens wurde die Verwendung semantischer Technologien vorgeschla-

gen. Operationen werden als **Referenzen auf semantische Spezifikationen in Funktions-Ontologien** (Abschnitt 6.3.3) modelliert, deren konkrete Implementierung zur Laufzeit im Anwendungskontext bestimmt wird (Abschnitt 6.3.4).

Die Beschreibung inhaltlicher Varianten (A1.3) erfolgte als **Modifikation der Struktur und Attribuierung des Strukturbaums** des Anwendungsmodells. Die Variantenbeschreibung ist dabei redundanzfrei, da sie lediglich Informationen des bestehenden Grundmodells modifiziert. In Abschnitt 6.5 wurde abschließend eine exemplarische Beschreibungssprache für Anwendungsmodelle und Variantenbeschreibungen in Form einer domänenspezifischen Sprache angegeben.

Die vorgeschlagene Modellierung löst die in Abschnitt 3.2 identifizierten offenen Problemstellungen in bestehenden Ansätzen. Sie beschreibt Benutzungsschnittstellen in einem **zentralen, technologieneutralen Artefakt** (Anwendungsmodell). Durch Hinzunahme eines weiteren Artefakts (differentielle Variantenbeschreibung) sind auf einfache Weise **redundanzfrei Varianten** spezifizierbar.

Einschränkungen und offene Fragestellungen. Der Fokus der Lösung liegt bewusst auf der Modellierung dialogbasierter Anwendungen. Daher bestehen ggf. Einschränkungen hinsichtlich der Modellierbarkeit von Anwendungsstrukturen, die über den dialogbasierten Charakter hinausgehen und die nicht im Fokus der Arbeit standen. Wie bereits in Abschnitt 5.7 in der Diskussion der Analyse angemerkt, kann die Vollständigkeit der Interaktionsmuster nicht garantiert werden. In zukünftigen Arbeiten werden ggf. Erweiterungen am Modell notwendig werden, die aus den untersuchten Anwendungen nicht ableitbar waren. Nach den gesammelten Erfahrungen führt dies lediglich zu einer Erweiterung der Lösung, nicht jedoch zur Widerlegung des dargestellten Ansatzes.

7. Modellierung von Kompositionen

Dieses Kapitel beschreibt einen neuartigen Ansatz zur Modellierung zusammengesetzter Benutzungsschnittstellen als Komposition. Er basiert auf der These, dass bestehende Anwendungsmodelle zu funktionsfähigen Kompositionen zusammengesetzt und ihrer Umgebung in Inhalt und Verhalten angepasst werden können, sofern Struktur und Verhalten der Komponenten bekannt ist. Der Ansatz erweitert den in Kapitel 6 dargestellten Modellierungsansatz um die Integration von bestehenden Anwendungsmodellen an beliebigen Stellen im Fragefluss der Komposition sowie deren Anpassung an die neue Nutzung.

Beitrag des Kapitels. Der Beitrag dieses Kapitels besteht in einer Methodik zur modellgetriebenen Entwicklung von Kompositionen bestehender Anwendungen. Die Neuerung liegt in der Anpassbarkeit wiederverwendeter Komponenten. Der Ansatz grenzt sich dadurch von bestehenden Lösungen ab, die lediglich eine Aggregation von Komponenten ermöglichen. Die Beiträge dieses Kapitels sind im Einzelnen:

- Ansatz zur Modellierung aus Komponenten zusammengesetzter Anwendungen
- Methode zur Anpassung von Inhalten und Verhalten der Komponenten in einer Komposition

Er erweitert bestehende Ansätze zur Wiederverwendung um Konzepte zur freien Kombination und Anpassung von Komponenten an einen veränderten Nutzungskontext.

7.1 Zielsetzung

In Abschnitt 2.4 wurden Kompositionen als Anwendungen vorgestellt, die an beliebigen Stellen im Fragefluss bestehende Komponenten einbinden. Die Komponenten sind selbständig lauffähig, müssen jedoch ggf. in Inhalt und Verhalten dem Nutzungskontext angepasst werden. Zudem interagieren sie ggf. mit anderen Komponenten in der Komposition. Hierdurch können aus bestehenden Komponenten erweiterte Anwendungen erstellt werden, die funktional über der Summe der einzelnen Komponenten liegen.

Das in Kapitel 6 vorgeschlagene Anwendungsmodell enthält alle Informationen, die zur Herleitung der Benutzungsschnittstelle erforderlich sind. Hierbei werden Anwendungsmodelle erstellt, die eine in sich vollständige *Komponente* beschreiben, aus der lauffähige Benutzungsschnittstellenvarianten generiert werden können. Sollen diese Anwendungsmodelle in Kompositionen wiederverwendet werden, stellt sich die in Abschnitt 1.3 gestellte Forschungsfrage **(FF.3)**, wie Anwendungsmodelle zu komplexeren Anwendungen kombiniert werden können.

Das Ziel dieses Kapitels ist die Erweiterung des bisherigen Ansatzes um die Modellierung von Anwendungen mit Bausteincharakter durch die **Wiederverwendung bestehender Anwendungsmodelle** sowie einer **Methodik zur Anpassung von Inhalten und Verhalten** der Komponenten im Rahmen einer Komposition.

7.2 Lösungsansatz

In Abschnitt 3.1 wurden die Anforderungen an die Modellierung von Kompositionen dargestellt. Sie bestanden in der freien Aggregierbarkeit von Komponenten **(A2.1)**, Anpassbarkeit der Komponenten an ihr Umfeld **(A2.2)** und vollständige Generierbarkeit der Komposition zu einer einheitlichen Benutzungsschnittstelle **(A2.3)**. Das in Abschnitt 3.3.3 vorgeschlagene Realisierungskonzept für Kompositionen besteht darin, vorhandene Anwendungsmodelle als Bausteine zu verwenden und daraus ein neues Anwendungsmodell zu aggregieren. Durch die Komposition erforderliche Änderungen an Inhalten und Verhalten der einzelnen Komponenten, werden als Variante der Aggregation aufgefasst und über eine Variantenbeschreibung modelliert.

Unter einer Komposition wird ein Anwendungsmodell gemäß Kapitel 6 verstanden, welches an definierten Stellen im Fragefluss weitere Anwendungsmodelle als Komponenten einbindet. Die Komponenten können wiederum Kompositionen sein. Daraus ergibt sich eine rekursive Komposition, die als Resultat wieder ein Anwendungsmodell ergibt. Die Komposition erfüllt dabei die Anforderung (A2.1). Die Anpassung der Inhalte und des Verhaltens der Komponenten (A2.2) erfolgt über den in Abschnitt 6.4 dargestellten Ansatz zur Modellierung von Varianten. Da es sich bei der Komposition um ein Anwendungsmodell handelt, kann dieser auch auf das aggregierte Modell angewendet werden.

Mit diesem Vorgehen ergibt sich die Erfüllung von Anforderung (A2.3) von selbst. Da eine Komposition selbst ein Anwendungsmodell gemäß Kapitel 6 ist, sind dieselben Generierungsverfahren anwendbar und finale Benutzungsschnittstellen für Kompositionen erzeugbar.

7.3 Modellierung von Kompositionen

Die Modellierung einer Komposition erfordert zwei Schritte, die in den folgenden Abschnitten dargestellt werden:

1. Aggregation von Komponenten zu einer Komposition
2. Modifikation von Inhalt und Verhalten von Komponenten

Abschnitt 7.3.1 stellt die Erweiterung des Anwendungsmodells aus Abschnitt 6.3 um die Integration von Komponenten dar. Abschnitt 7.3.2 zeigt auf, wie Elemente innerhalb einer Komposition referenziert werden können. Dies bildet die Grundlage zur Anpassung des Inhalts und Verhaltens einer Komposition, welche in Abschnitt 7.3.3 beschrieben wird.

7.3.1 Aggregation von Komponenten

In Abschnitt 6.3.1 wurden Anwendungsmodelle mittels attribuierter Strukturbäume modelliert, deren innere Knoten Gruppen von Daten und deren Blätter Datenelemente repräsentieren. Die Komposition von Anwendungen kann als Integration der Strukturbäume von Komponenten in den Strukturbaum der Komposition aufgefasst werden. In Abbildung 7.1 ist der Strukturbaum einer Komposition dargestellt. Dieser bindet an definierten Knoten (*Integrationsknoten*) die Strukturbäume weiterer Komponenten ein.

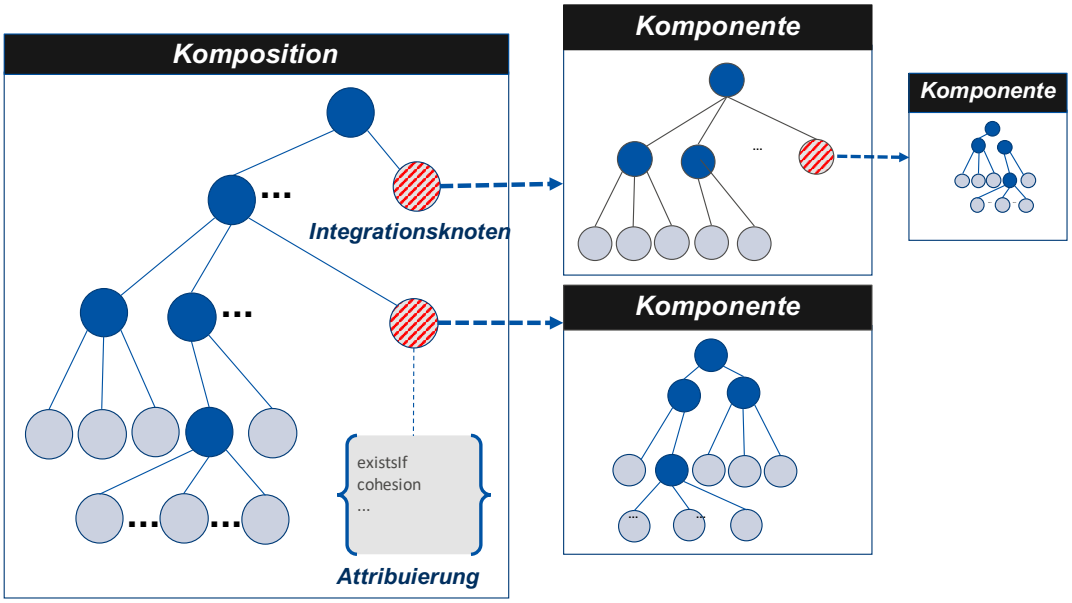


Abbildung 7.1. Strukturbaum einer Komposition und deren Komponenten

Integrationsknoten stellen Platzhalter dar, welche durch die Wurzelknoten der referenzierten Komponenten ersetzt werden. Ein Integrationsknoten ist ein spezieller Gruppenknoten, der den Teilbaum der Komponente repräsentiert. Die Attribuierung eines Integrationsknotens entspricht ebenfalls der einer Gruppe. So können für integrierte Komponenten z.B. *Existenzbedingungen* oder die *äußere Kohäsion* angegeben werden, die für die Komponente in der Komposition gelten sollen.

Das in Abschnitt 6.3.1 vorgestellte Metamodell kann auf einfache Weise zur Modellierung von Kompositionen angepasst werden. Abbildung 7.2 zeigt das erweiterte Metamodell. Für Integrationsknoten wird eine Knotenart *DataComponent* eingeführt, die von *DataGroup* abgeleitet ist. Die **Spezifikation der einzubindenden Komponente** erfolgt über deren eindeutige URI (*componentURI*). Optional kann eine bestimmte Variante angegeben werden. Dies erfolgt über die Angabe der URI der Variante (*variantURI*).

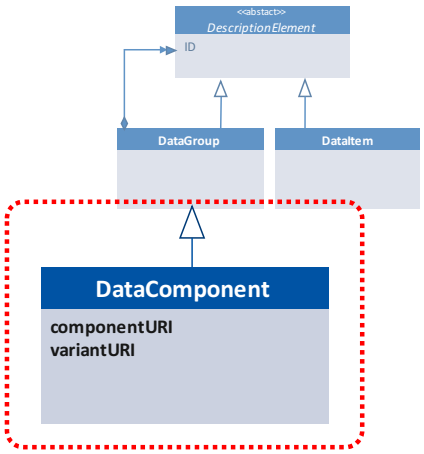


Abbildung 7.2. Erweiterung des Metamodells um Komponenten

Die Aggregation des Anwendungsmodells der Komposition erfolgt in zwei Schritten. Im ersten Schritt wir basierend auf den Informationen der Integrationsknoten für jede referenzierte Komponente ein Strukturbaum erzeugt. Die resultierenden Strukturbäume werden im zweiten Schritt in den Strukturbaum der Komposition eingefügt.

7.3.2 Angabe von Modellelement-Referenzen in Kompositionen

Anwendungsmodelle und Variantenbeschreibungen enthalten Referenzen auf Elemente im Strukturbaum der Anwendung. Diese werden zur Laufzeit zur Referenzierung von Daten im Anwendungskontext benötigt. Zudem werden sie in Variantenbeschreibungen zur Identifikation zu modifizierender Modellelemente verwendet (vgl. Abschnitt 6.3.3). Die Identifikation eines Elements erfolgt dabei über die Angabe eines Pfades, der seine Position im Strukturbaum bezeichnet.

In Kompositionen können Referenzen auf Modellelemente weiterhin über den Pfad im Strukturbaum angegeben werden, da jede Komponente eine eindeutigen Position im Strukturbaum der Komposition besitzt.

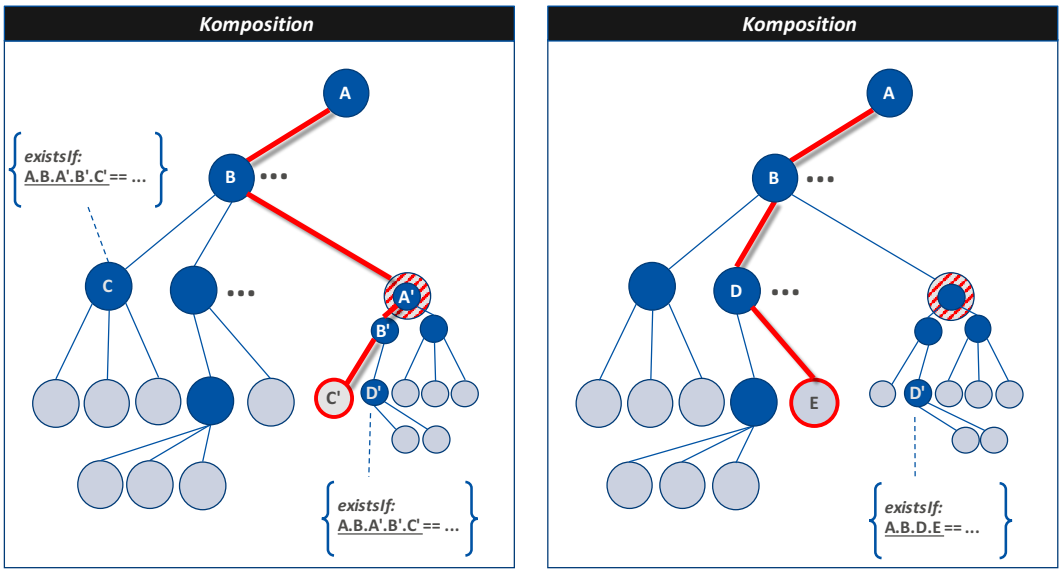


Abbildung 7.3. Referenzierung von Modellelementen in Kompositionen

In Abbildung 7.3 (links) referenziert der Knoten C in einer Existenzbedingung den Knoten C' einer Komponente. Der Pfad von C' innerhalb der Komponente lautet $A'.B'.C'$. Zur Referenzierung des Knotens C' im Rahmen der Komposition muss dem Pfad die Position des Integrationsknotens vorangestellt werden. Der eindeutige Pfad lautet dann $A.B.A'.B'.C'$. Analog hierzu kann eine Komponente auf Elemente der Komposition und anderer Komponenten zugreifen. In Abbildung 7.3 (rechts) referenziert der Knoten D' in einer Existenzbedingung den Knoten E der Komposition. Der Pfad dieses Knotens lautet $A.B.D.E$.

Die Verwendung von Pfadangaben ausgehend vom Wurzelknoten der Komposition gestattet **eine Verzahnung der Komponenten untereinander**. Da Kompositionen Inhalte von Komponenten referenzieren können, kann die Komposition auf den Zustand der Komponenten reagieren. Da auch Komponenten damit in der Lage sind, auf Informationen umliegender Modellelemente zuzugreifen, können **komponentenübergreifende Abhängigkeiten**

modelliert werden. Eine Komponente kann auf den Zustand der umliegenden Komposition und deren Komponenten reagieren. Dies ist die Voraussetzung für die im folgenden Abschnitt beschriebene Anpassung von Komponenten einer Komposition.

7.3.3 Anpassung von Komponenten in Kompositionen

Um das Zusammenspiel der Komponenten einer Komposition steuern zu können, müssen die eingebundenen Komponenten hinsichtlich ihres Inhalts, ihrer Elementeigenschaften und ihres Verhaltens an ihr Umfeld angepasst werden (A2.2). Der in Abschnitt 6.4 beschriebene Ansatz zur Modellierung von Varianten von Anwendungsbeschreibungen beinhaltet bereits die hierfür erforderlichen Möglichkeiten zur

- Auswahl relevanter Elemente durch Elementregeln (Abschnitt 6.4.1)
- Modifikation von Elementeigenschaften durch Modifikatoren (Abschnitt 6.4.2)

Die Voraussetzung zur Anwendung des Ansatzes ist die Referenzierbarkeit von Elementen des Strukturbaums. Diese wurde im vorangegangenen Abschnitt 7.3.2 für Kompositionen dargestellt. Damit können ggf. erforderliche Anpassungen für Kompositionen erstellt werden. Für die Anpassungen wird eine Variantenbeschreibung verwendet, welche Modifikationen an den Inhalten des **aggregierten Strukturbaums der Komposition** beschreibt. In Abbildung 7.4 ist dies exemplarisch dargestellt.

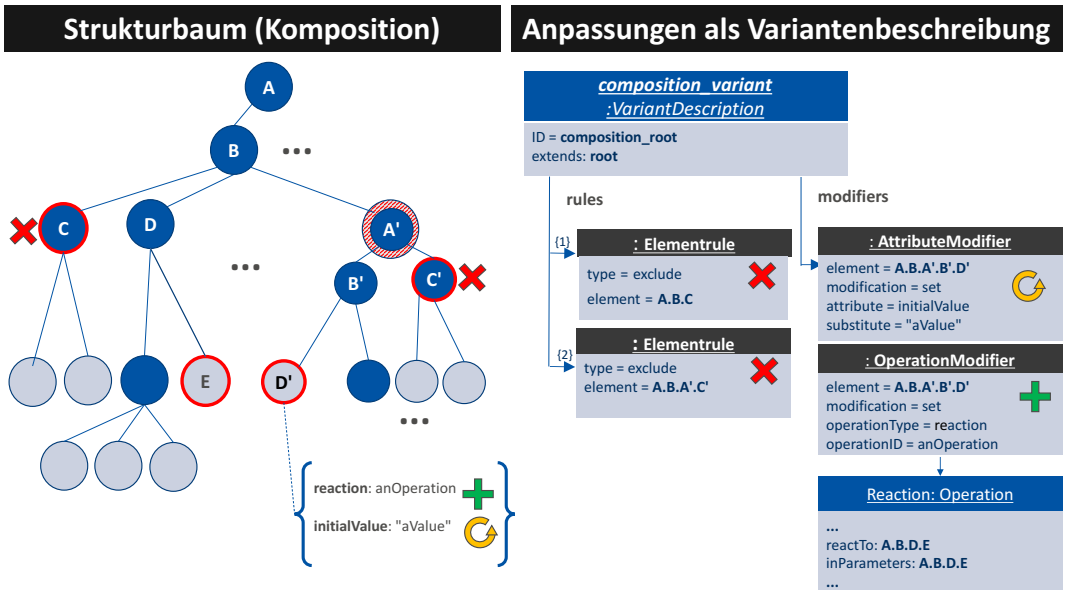


Abbildung 7.4. Anpassung der Komponenten einer Komposition

Die **inhaltlichen Änderungen** einer Komponente werden über *Elementregeln* beschrieben. In Abbildung 7.4 werden exemplarisch die Knoten C und C' in der Komposition ausgeschlossen. C' ist dabei das Element einer Komponente.

Änderungen an Elementeigenschaften und des Verhaltens erfolgen durch Angabe von *Elementmodifikatoren*, die die Konten des Strukturbaums passend zum Anwendungsfall der Komposition modifizieren. In Abbildung 7.4 wird exemplarisch dem Knoten D' über eine Attributmodifikation ein neuer Initialwert zugeordnet. Über eine Operationsmodifikation

wird zudem eine Reaktion zugewiesen. Das Element D' reagiert dabei auf die Änderung des Feldes $A.B.D.E$, welches außerhalb des Komponente liegt.

7.4 Beispiel einer Komposition

Zur Illustration der Modellierung von Kompositionen dient das Beispiel einer Reisebuchung, die mit einer Reiseversicherung kombiniert wird (Abschnitt 2.4.1). Hierfür wird angenommen, dass die Reiseversicherungs-Komponente für mehrerer Produktkomponenten konzipiert wurde (z.B. Transportkosten, Gepäck, Übernachtungskosten). In Rahmen der kombinierten Anwendung soll diese wiederverwendet werden, jedoch nur die Transportkosten versicherbar sein.

Listing 7.1 stellt einen Ausschnitt der Komposition als Rahmenanwendung dar. Die Rahmenanwendung erfasst dabei zuerst die Kundendaten ①. Die Komposition referenziert eine Flugbuchungs- und eine Reiseversicherungs-Komponente ②③. Zudem kann der Kunde nach Angabe der Flugdaten bestimmen, ob er eine Reiseversicherung wünscht ④. Die Versicherungskomponente soll nur angezeigt werden, wenn der Kunde diese Option wählt ⑤.

```
   data_description : "fly_safe" {
      ...
❶  group: "customerdata" {
      ...
      firstname :{...}
      lastname : {...}
    } ...
❷  component : "flight" {
      uri    ="http://mimesis.solutions/bookers/flightbooking/v1#"
      variant  ="http://mimesis.solutions/bookers/flightbooking/variants/...
          ↪ v1#customer_minimal"
    }
    ...
❹  insurancerequested: {type="boolean" initialValue="true"}
    ...
❸  component : "insurance" {
❺       existsIf="fly_safe.<path>.insuranerequested == true" }
    {
      uri    ="http://mimesis.solutions/insurance/travel/v1#"
    }
  }
```

Listing 7.1: Komposition der Flugbuchung

Listing 7.2 stellt einen Ausschnitt der Anwendungsmodelle der referenzierten Komponenten dar ①,②. Beide Komponenten beinhalten Kundendaten ③, die jedoch nicht angezeigt sondern mit den Angaben der Rahmenanwendung befüllt werden sollen. Die Versicherungskomponente enthält eine M-aus-N-Auswahl zur Angabe der zu versichernden Module (*productcomponents*, ④). Das Feld ist in der Komposition nicht sichtbar und mit *transport* vorbelegt.

Die Anpassung der Komponenten in der Komposition erfolgt in einer Variantenbeschreibung (s. Listing 7.3).

```
❶  data_description : "flight_booking" {
     ...
❸   group: "customerdata" {
       firstname :{}
       lastname : {}
     } ...
   }

❷ data_description : "travel_insurance" {
❸  group: "customerdata" {
       firstname :{}
       lastname : {}
     } ...
❹   productcomponents :{restrictedTo="transport / luggage / overnightstays"
                        cardinality="*"}
    ...
     group: "transportdata" {...}
     group: "luggagedata" {...}
     group: "overnightstaysdata" {...}
   }
```

Listing 7.2: Komponenten der Anwendung

```
  variant : "root" { ...
❶   exclude: "fly_safe.insurance.luggagedata"
    exclude: "fly_safe.insurance.overnightstaysdata"

     // Modifikation der Versicherungskomponente
❷   modify : { type  : "attribute"
               element : "fly_safe.insurance.customerdata.firstname"
               attribute : "hidden"
               substitute : "true"}
     modify : { type  : "attribute"
               element  : "fly_safe.insurance.customerdata.lastname"
               attribute  : "hidden"
               substitute : "true"}
❸   modify : { type  : "attribute"
               element : "fly_safe.insurance.customerdata.firstname"
               attribute : "reference"
               substitute : "fly_safe.customerdata.firstname"}
     modify : { type : "attribute"
               element : "fly_safe.insurance.customerdata.lastname"
               attribute : "reference"
               substitute : "fly_safe.customerdata.lastname"}
❹   modify : { type : "attribute"
               element : "fly_safe.insurance.productcomponents"
               attribute : "hidden"
               substitute : "true"}
     modify : { type : "attribute"
               element : "fly_safe.insurance.productcomponents"
               attribute : "initialValue"
               substitute : "transport"}
  }
```

Listing 7.3: Variantenbeschreibung der Komposition

In der Variantenbeschreibung sind die Modifikationen der Versicherungskomponente aufgeführt. Die Modifikationen für die Flugbuchung (z.B. ausblenden der Kundendaten) erfolgen analog und sind daher nicht dargestellt.

In ① werden Teile der Versicherungskomponente entfernt, die in der Komposition nicht benötigt werden. Insbesondere sind dies Fragen zu Produktkomponenten, die in der Komposition nicht wählbar sind. Zudem werden die Elemente der Kundendaten dauerhaft ausgeblendet ② und deren Inhalt über eine Elementreferenz (*reference*) mit entsprechenden Angaben in der Rahmenanwendung verknüpft ③. Die Auswahl der Produktkomponenten in der Versicherungskomponente (*productcomponents*) wird ebenfalls ausgeblendet und mit dem Initialwert *transport* versehen ④.

In Anhang A.8.3 bis A.8.5 ist zur weiteren Untersuchung eine Reisebuchung als komplexeres Beispiel dargestellt. Hier werden multiple Komponenten eingebunden und deren Inhalte modifiziert. Zu interaktiven Untersuchung einer lauffähigen Variante s. die Verweise in Anhang A.7.

7.5 Diskussion der Ergebnisse

Ziel des Kapitels war die Darstellung einer Methode zur Wiederverwendung und Anpassung vorhandener Anwendungsmodelle in Kompositionen. Im Verlauf des Kapitels wurde gezeigt, wie aus bestehenden Komponenten ein neues Anwendungsmodell aggregiert und Komponenten hinsichtlich Inhalt und Verhalten angepasst werden können. Zudem wurde das Metamodell für die Verwendung von Komponenten erweitert. Als Resultat der Komposition entsteht ein vollständiges Anwendungsmodell gemäß Abschnitt 6.3.1 und erfüllt damit Anforderung (A2.3). Komponenten können an beliebigen Stellen im Fragefluss einer Komposition eingefügt (A2.1) und deren Inhalt und Verhalten der neuen Nutzung angepasst werden (A2.2).

Der Vorteil gegenüber bestehenden Ansätzen liegt insbesondere in der Anpassbarkeit der Komponenten und deren Verknüpfung mit anderen Komponenten in der Komposition. Während in bestehenden Ansätzen lediglich eine Aggregation möglich ist, besitzt der Ersteller einer Komposition im vorgeschlagenen Ansatz eine weitreichende Kontrolle über eingebundene Komponenten und kann deren Zusammenspiel beeinflussen. Das Ergebnis ist eine Benutzungsschnittstelle, die funktional über die Komponenten hinausgehen kann. Die Komposition ist in ihrer Gesamtheit generierbar und erscheint damit “wie aus einem Guss”.

Gleichzeitig ist diese **Flexibilität ein potentieller Nachteil** des Ansatzes: es ist grundsätzlich möglich, eine eingebundene Komponente soweit zu modifizieren, dass die ursprüngliche Funktion verloren geht. Durch die umfassende Modifizierbarkeit des Inhalts und Verhaltens kann nicht sichergestellt werden, dass die *Semantik der Komponenten* erhalten bleibt. Der Ersteller der Komposition ist für den Erhalt der Semantik verantwortlich. Im Rahmen der Arbeit wurde dieser Aspekt nicht näher betrachtet. Zukünftige Arbeiten können sich mit sinnvollen Einschränkungen der Modifizierbarkeit befassen, welche die Semantik einer eingebundenen Komponente erhalten und deren konsistente Nutzung in Kompositionen sicherstellen.

8. Verfahren zur Generierung von Benutzungsschnittstellen, Varianten und Kompositionen

Das Kapitel beschreibt ein Verfahren zur vollständig automatisierten Generierung von Benutzungsschnittstellenvarianten aus datenzentrierten Anwendungsmodellen für multiple Nutzungs-, Interaktions- und Technologiekontexte. Die Grundlage hierfür bildet das in Kapitel 6 und 7 entwickelte Anwendungsmodell und zugehörige Variantenbeschreibungen. Es wird dargestellt, wie aus diesen technologieneutralen Modellen Kompositionen und Varianten abgeleitet und schrittweise finale Benutzungsschnittstellen generiert werden, die auf einen konkreten Interaktions- und Technologiekontext zugeschnitten sind. Das Verfahren setzt dabei auf dem CAMELEON-Framework auf und erweitert dieses um Verfahren zur Ableitung von Kompositionen und Varianten.

Beitrag des Kapitels. Der Beitrag des Kapitels besteht in einem Verfahren zur Generierung von Benutzungsschnittstellenvarianten aus technologieneutralen, datenzentrierten Anwendungsmodellen. Es erweitert bestehende Ansätze um Verfahren zur vollständig automatisierten Erzeugung inhaltlicher Varianten und Kompositionen.

8.1 Zielsetzung

Das Ziel dieses Kapitels besteht in der Darstellung eines Verfahrens zur Herleitung finaler Benutzungsschnittstellen aus den in den Kapiteln 6 bis 7 beschriebenen Modellen (**Anwendungsmodell**, **Variantenbeschreibung** und **Kompositionen**). Es beantwortet damit die Forschungsfrage **(FF.4)** (vgl. Abschnitt 1.3).

Die Anforderungen an das Verfahren wurden in Abschnitt 3.1, Tabelle 3.1 spezifiziert. Danach muss das Verfahren aus den Modellen automatisiert inhaltliche Varianten und Kompositionen erstellen **(A3.1)** und daraus technische Varianten für konkrete Interaktions- und Technologiekontexte erzeugen **(A3.2)**. Das Ergebnis muss in einer Laufzeitumgebung ausführbar sein **(A3.3)**. Die erzeugten Benutzungsschnittstellen müssen dabei die Eigenschaften aufweisen, die für dialogbasierte Benutzungsschnittstellen in den Abschnitten 5.3-5.6, Tabellen 5.1-5.4 identifiziert wurden (IR.1-IR.17).

Die Darstellung des Verfahrens in den folgenden Abschnitten gliedert sich wie folgt: In **Abschnitt 8.2** wird der Lösungsansatz im Überblick dargestellt. In **Abschnitt 8.3** wird die Herleitung von Anwendungsmodellen für inhaltliche Varianten und Kompositionen dargestellt. **Abschnitt 8.4** beschreibt das Verfahren zur Generierung der finalen Benutzungsschnittstelle. **Abschnitt 8.5** zeigt die Aufgaben einer konkreten Laufzeitumgebung zur Darstellung und Ausführung auf. **Abschnitt 8.6** diskutiert die Ergebnisse und gibt einen Ausblick auf zukünftige Arbeiten.

8.2 Lösungsansatz

Zur Umsetzung der Anforderungen (A3.1) und (A3.2) muss ein Verfahren zur Generierung der Benutzungsschnittstelle aus dem Anwendungsmodell und einer Variantenbeschreibung folgende Schritte durchführen:

1. Aggregation referenzierter Komponenten zu einem Gesamtmodell
2. Ableitung der inhaltlichen Variante aus Gesamtmodell und Variantenbeschreibung
3. Ableitung eines abstrakten Benutzungsschnittstellenmodells
4. Ableitung einer finalen Benutzungsschnittstelle für einen spezifischen Interaktions- und Technologiekontext

Das Resultat wird abschließend an eine Laufzeitumgebung übergeben, welche die Benutzungsschnittstelle darstellt und ausführt. Der Gesamtprozess kann in drei Phasen unterteilt werden, die in Abbildung 8.1 dargestellt sind und in den folgenden Abschnitten im Detail erläutert werden:

- Variantentransformation (s. Abschnitt 8.3)
- Benutzungsschnittstellentransformation (s. Abschnitt 8.4)
- Darstellung und Ausführung in der Laufzeitumgebung (s. Abschnitt 8.5)

In jeder Phase wird das Modell sukzessive konkretisiert und in die ausführbare Form für eine spezifische Laufzeitumgebung transformiert. Bei der **Variantentransformation** wird aus dem Anwendungsmodell und der Variantenbeschreibung ein variantenspezifisches Anwendungsmodell erzeugt (**Variantenmodell**). Hierzu werden zuerst ggf. referenzierte Komponenten einer Komposition zu einem vollständigen Anwendungsmodell aggregiert. Im nächsten Schritt werden die in der Variantenbeschreibung angegebenen Regeln und Modifikationen auf das aggregierte Anwendungsmodell angewendet.

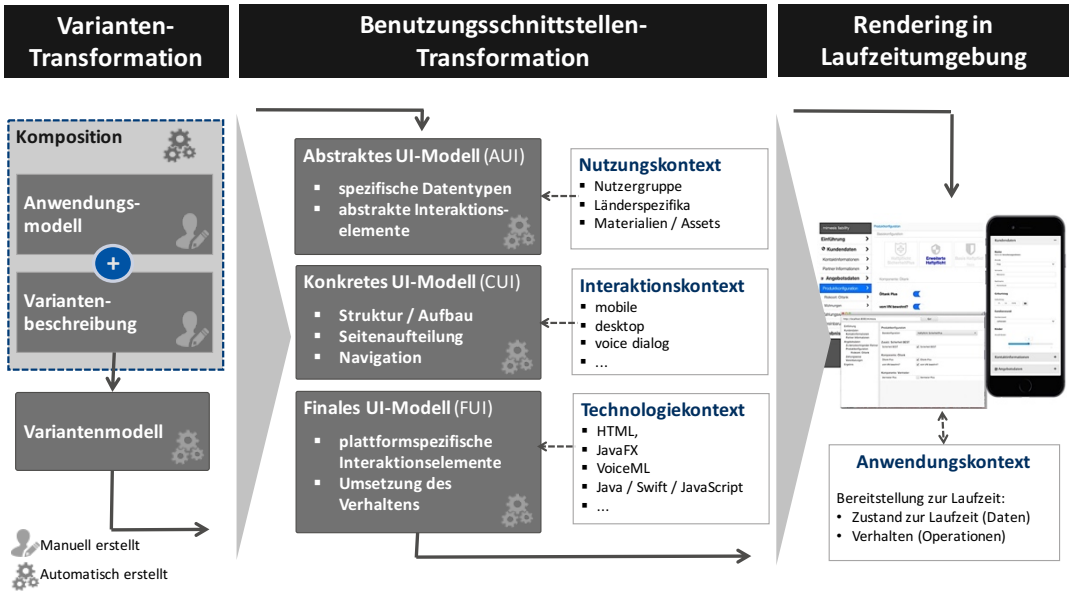


Abbildung 8.1. Phasen und Prozessschritte zur Herleitung der Benutzungsschnittstellenvarianten

Die **Benutzungsschnittstellentransformation** überführt das Variantenmodell in ein **Benutzungsschnittstellenmodell** für eine technische Umgebung. Hierbei werden zuerst die Datenelemente des Anwendungsmodells auf Interaktionselemente abgebildet und die Struktur erzeugt (Gruppierung, Verteilung auf Darstellungseinheiten). Das resultierende Benutzungsschnittstellenmodell wird anschließend in ein finales Modell für eine konkrete Lauf-

zeitumgebung überführt. Die Transformation benötigt hierzu Informationen zum Nutzungs-, Interaktions- und Technologiekontext der Anwendung.

Die **Darstellung und Ausführung** erfolgt in der Laufzeitumgebung. Hierbei werden aus dem finalen Benutzungsschnittstellenmodell konkrete Interaktionselemente zur Laufzeit erzeugt (*Rendering der Benutzungsschnittstelle*). Die Laufzeitumgebung steuert dabei die Ausführung der Anwendung und stellt den **Anwendungskontext** bereit, der den Zustand der Anwendung verwaltet, sowie die Implementierung der benötigten Operationen zur Verfügung stellt.

8.3 Variantentransformation

Das Anwendungsmodell enthält den **attribuierten Strukturbaum** der Anwendung, der ggf. weitere Anwendungsmodelle als Komponenten referenziert. Die Variantenbeschreibung enthält **Elementregeln** und **Elementmodifikatoren**, welche Änderungen am Strukturbaum und dessen Knoten für eine spezifische Variante bestimmen. Die Herleitung des Variantenmodells erfolgt in zwei Schritten:

1. Aggregation der Komponenten zu einem Gesamtmodell
2. Erzeugung des konkreten Variantenmodells

Im ersten Schritt werden die Referenzen auf die Komponentenmodelle aufgelöst und daraus ein komplettes Anwendungsmodell erstellt. Darauf aufbauend erfolgt im zweiten Schritt die Herleitung des Variantenmodells durch die Anwendung der in der Variantenbeschreibung angegebenen Elementregeln und -modifikatoren.

Aggregation der Komponenten zu einem Gesamtmodell

Die Erstellung des Gesamtmodells kann als eine Folge von Einfüge-Operationen aufgefasst werden, die den Strukturbaum der Anwendung erweitern. Im Strukturbaum vorhandene *Integrationsknoten* werden dabei durch die Strukturbäume der Komponenten ersetzt (vgl. Abschnitt 7.3.1). Algorithmus 1 zeigt das Vorgehen als Pseudocode.

Algorithmus 1 Aggregation des Gesamtmodells

```
1:  procedure composeApplicationModel(AM)
2:      inputs
3:          Anwendungsmodell (AM),
4:      result
5:      Aggregiertes Anwendungsmodell (CM)
6:      begin
7:          CM ← AM
8:          for amNode in CM do
9:              if amNode ofType integrationNode then
10:                 componentModel ← getComponentModel(amNode.URI)
11:                 componentModel.adjustContextReferences (amNode.path)
12:                 insertComponentmodel(amNode,componentModel)
13:     return CM
```

Als Eingabe für das Verfahren dient der Strukturbaum einer Anwendung, dessen Knoten traversiert werden. Handelt es sich bei einem besuchten Knoten um einen *Integrationsknoten* (s. Abschnitt 7.3.1), wird anhand der angegebenen URI ein Strukturbaum der Komponente erzeugt (*getComponentModel()*), der den Integrationsknoten ersetzt. Da sich aufgrund der Integration in den umgebenden Strukturbaum der Komposition Referenzen auf den Anwendungskontext ändern, werden diese im Komponentenstrukturbaum gemäß der neuen Position im Gesamtstrukturbaum angepasst (*adjustContextReferences()*). Abschließend wird der Inhalt des Integrationsknotens durch den modifizierten Komponentenstrukturbaum ersetzt (*insertComponentModel()*).

Das Resultat ist ein **vollständiger Strukturbaum der Anwendung**, der sämtliche Teilbäume der Komponenten enthält und die Grundlage für die Erzeugung des Variantenmodells bildet.

Erzeugung des konkreten Variantenmodells

Die Erzeugung des Variantenmodells erfolgt durch Anwendung der in der Variantenbeschreibung enthaltenen Regeln und Modifikatoren auf das Anwendungsmodell. Eine **Elementregel** kann dabei als Operation auf dem Strukturbaum aufgefasst werden, die einen Knoten des Baums aus dem Strukturbaum entfernt bzw. hinzufügt. Ein **Elementmodifikator** hingegen beschreibt die Änderung von Eigenschaften eines einzelnen Knotens. Sie ist daher eine Operation auf *Attributen* eines Strukturbaumknotens.

Algorithmus 2 zeigt die Schritte zur Erzeugung des Variantenmodells als Pseudocode. Die Eingabeparameter für den Prozess sind das Anwendungsmodell (AM) und die Variantenbeschreibung (VD).

Algorithmus 2 Ableitung eines Variantenmodells

```
 1: procedure DERIVEVARIANTMODEL(AM, VD)
 2:     inputs
 3:         Anwendungsmodell (AM),
 4:         Variantenbeschreibung (VD)
 5:         → VD.elementrules: geordnete Liste der Elementregeln
 6:         → VD.elementmodifiers: geordnete Liste der Elementmodifikatoren
 7:     result
 8:     Variantenmodell (VM)
 9:     begin
10:         // initialisiere Variantenmodell
11:         AM' ← AM
12:         for r in VD.elementrules do
13:             applyElementrule (r, AM')
14:         for m in VD.elementmodifiers do
15:             applyElementmodifier (m, AM')
16:         VM ← AM'.removeMarkedElements ()
17:     return VM
```

Zuerst werden die Elementregeln der Variantenbeschreibung auf den Strukturbaum angewendet. Hierzu werden alle Baumknoten markiert, die gemäß den Regeln nicht in der Variante enthalten sein sollen (*applyElementrule()*). Anschließend werden alle Elementmodifikationen der Variantenbeschreibung angewendet (*applyElementmodifier()*). Die Modifikatoren verändern dabei die Eigenschaften des Baumknotens, indem die Knotenattribute (z.B.

Restriktionen, Operationen) modifiziert werden. Abschließend wird aus dem modifizierten Strukturbaum (AM') das Variantenmodell erzeugt (VM), indem alle zur Exklusion markierten Knoten aus dem Strukturbaum entfernt werden.

In Anhang A.4 sind die Operationen zur Anwendung der Elementregeln und Elementmodifikationen beschrieben. Anhang A.4.1 zeigt den Markierungsalgorithmus zur Exklusion von Elementen im Detail. In Anhang A.4.2 wird die Modifikation von Strukturbaumknoten für Attribut- und Operationsmodifikationen dargestellt.

Das Resultat ist eine **Variante des ursprünglichen Anwendungsmodells** (VM), welche lediglich die in der Variantenbeschreibung festgelegten Elemente enthält. Die Eigenschaften der verbleibenden Elemente wurden basierend auf den Modifikatoren angepasst.

8.4 Benutzungsschnittstellentransformation

Aus dem Variantenmodell wird in dieser Phase ein Modell erzeugt, das die Anwendung (Struktur, Interaktionselemente, Verhalten) aus Benutzungsschnittstellen-Sicht beschreibt. Die Transformation erzeugt dabei ein Modell der Benutzungsschnittstelle, welches auf einen **spezifischen Nutzungs-, Interaktions- und Technologiekontext** zugeschnitten ist. Es enthält alle Informationen, die zur Ausführung in einer konkreten Laufzeitumgebung benötigt werden. Die Transformation umfasst folgende Aktivitäten:

- Abbildung der Datenelemente und des Verhaltens auf Standard-Interaktionselemente
- Strukturierung durch Gruppierung und Verteilung der Elemente auf Seiten
- Erzeugung eines plattformspezifischen Modells mit dort verfügbaren technischen Interaktionselementen

Das im Folgenden beschriebene Verfahren basiert auf dem im CAMELEON-Framework von Calvary et al. ([25]) vorgeschlagenen Vorgehen (vgl. Abschnitt 2.6.3). Die Transformation vom Ausgangsmodell bis zur finalen Benutzungsschnittstelle erfolgt in drei Schritten über die Ableitung eines Modells für eine

- abstrakte Benutzungsschnittstelle (*Abstract User Interface* - AUI)
- konkrete Benutzungsschnittstelle (*Concrete User Interface* - CUI)
- finale Benutzungsschnittstelle (*Final User Interface* - FUI)

Abbildung 8.2 stellt die Transformationsschritte dar, wie sie für diese Arbeit konkretisiert wurden. Sie zeigt die durchzuführenden Aktivitäten und die Quellen, aus denen die Informationen zu deren Durchführung stammen.

Die **AUI-Transformation** überführt das Variantenmodell (Abschnitt 8.3) in ein abstraktes Benutzungsschnittstellenmodell. Hierzu werden die darin enthaltenen Typinformationen (Typ, Restriktionen, semantische Annotationen) ausgewertet und entsprechende Standard-Interaktionselemente für die Elemente des Anwendungsmodells bestimmt. Der **Nutzungskontext** beeinflusst dabei die Auswahl der Interaktionselemente u.a. durch länder- bzw. nutzergruppenspezifische Eingaben mit entsprechenden Validierungen (z.B. die Eingabe und Validierung einer länderspezifischen Postleitzahl).

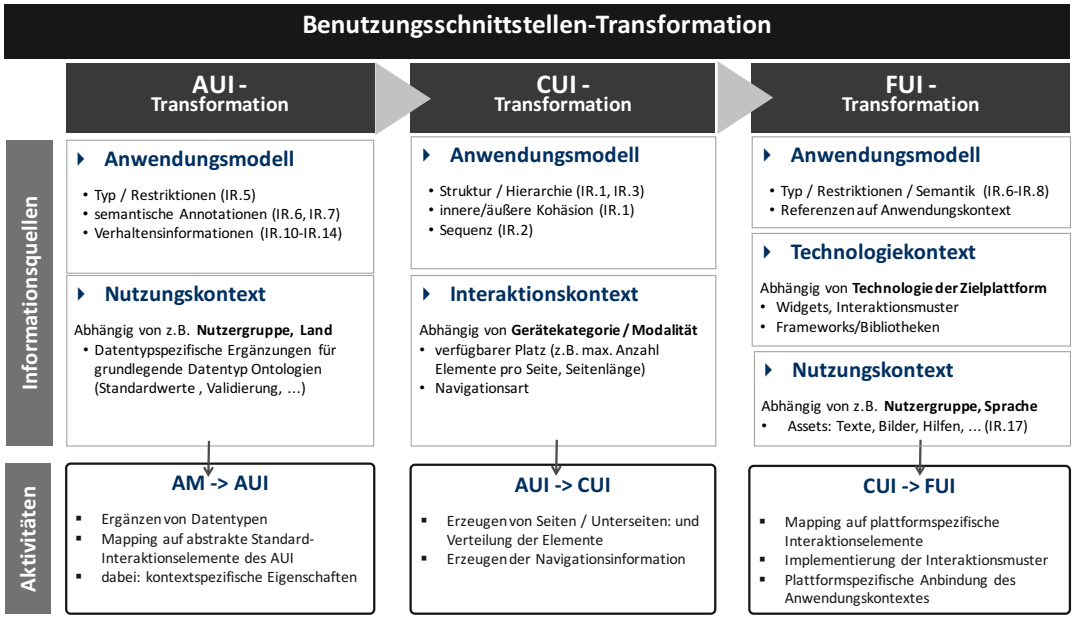


Abbildung 8.2. Schritte der Benutzungsschnittstellentransformation

Die **CUI-Transformation** leitet aus dem AUI ein konkretes Benutzungsschnittstellenmodell für einen spezifischen Interaktionskontext her. Der Interaktionskontext bestimmt dabei Rahmenbedingungen, die für die Benutzungsschnittstelle in diesem Kontext gelten. In diesem Schritt wird der Strukturbaum angepasst, sodass er die Fragen gemäß geltender Einschränkungen der gewählten Interaktionsform (z.B. Gerätekategorie) strukturiert. Hierzu werden insbesondere die Interaktionselemente auf Darstellungseinheiten verteilt. Aus dieser Anpassung resultiert zudem die Navigationsstruktur, die dem Nutzer präsentiert wird.

Das Resultat der ersten beiden Schritte ist ein Benutzungsschnittstellenmodell, aus dem in der **FUI-Transformation** die finale Benutzungsschnittstelle für einen konkreten Technologiekontext hergeleitet wird. Hierbei werden Dialogseiten und konkrete (ggf. applikationsspezifische) Interaktionselemente für die auf der Zielplattform vorhandenen Technologie erzeugt. Neben der Darstellung werden dabei auch das Verhalten (z.B. zeigen/verbergen von Bereichen, Anstoßen von Operationen) mit den Mitteln die Zieltechnologe beschrieben, sodass diese von der Ablaufumgebung interpretiert und ausgeführt werden können. Das Resultat ist eine Benutzungsschnittstellenbeschreibung, die einer konkreten plattformspezifischen Ablaufumgebung zur Darstellung übergeben und ausgeführt werden kann.

Im Folgenden wird zunächst der Aufbau und die Inhalte des oben genannten Benutzungsschnittstellenmodells dargestellt, welches die Grundlage der dargestellten Transformationen bildet. Anschließend werden die Schritte zur Transformation des Anwendungs- in das Benutzungsschnittstellenmodell beschrieben, sowie die Überführung in die finale Benutzungsschnittstelle dargestellt.

8.4.1 Zugrundegelegtes Benutzungsschnittstellenmodell

Das Benutzungsschnittstellenmodell beschreibt die Struktur, die Interaktionselemente und das Verhalten einer Benutzungsschnittstelle für eine dialogbasierte Anwendung. In Kapi-

tel 5 wurde dargestellt, dass diese über eine hierarchische Struktur aus Darstellungseinheiten, Untereinheiten, verschachtelten Gruppen und typisierten Interaktionselementen beschrieben werden kann. Im Verlauf der Analyse wurde ein Modell für Benutzungsschnittstellen für dialogbasierte Anwendungen abgeleitet, welches die Eigenschaften der untersuchten Benutzungsschnittstellen abbildet [98]. Die Ergebnisse dienten als Grundlage für die Herleitung des in Kapitel 6 beschriebenen datenzentrierten Anwendungsmodells. Das Benutzungsschnittstellenmodell ist daher hinsichtlich der enthaltenen Informationen dem Anwendungsmodell ähnlich.

Im Folgenden wird ein Überblick über die Elemente des Modells gegeben. In Anhang A.2.3 werden die Modellelemente und deren Attribute näher beschrieben. Anhang A.3.3 stellt eine Notation zur Beschreibung des Benutzungsschnittstellenmodells als DSL im JSON-Format vor. Diese wird in den folgenden Darstellungen in den Beispielen verwendet.

Struktur und Aufbau. Abbildung 8.3 zeigt exemplarisch die Elemente zu Struktur und den Aufbau für einer *grafischen Benutzungsschnittstelle*, wie sie in Abschnitt 5.3 beschrieben wurde. Eine Benutzungsschnittstelle beinhaltet Darstellungseinheiten (*DisplayUnits*), die wiederum Untereinheiten besitzen können (Abbildung 8.3, links). In der grafischen Benutzungsschnittstelle werden diese zu Seiten und Unterseiten, die über eine Navigation auswählbar sind. Jede Darstellungseinheit beinhaltet wiederum Darstellungselemente (*DisplayElements*), die aus Gruppen (*Groups*) und Interaktionselementen (*Fields*) zusammengesetzt sind. Abbildung 8.3 zeigt dies am Beispiel einer Seite der Oberfläche. Hier sind die grafischen Interaktionselemente (mitte) und deren Strukturierung über Darstellungselemente als Gruppen, Untergruppen und Eingabefelder dargestellt (rechts).

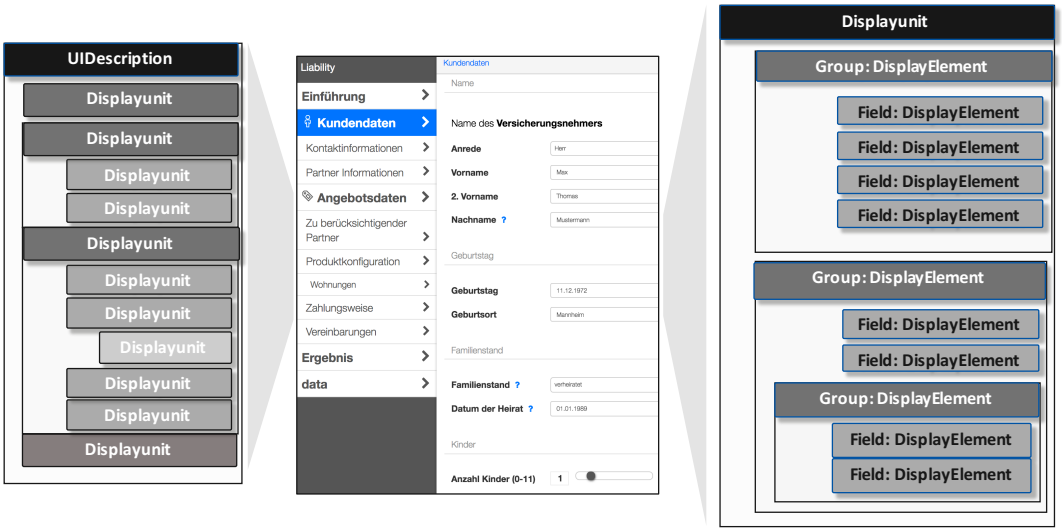


Abbildung 8.3. Beispiel für die Strukturelemente des Benutzungsschnittstellenmodells

Abbildung 8.4 zeigt das Benutzungsschnittstellenmodell als UML-Diagramm, welches diese Struktur beschreibt. Eine Benutzungsschnittstellenbeschreibung (*UIDescription*) besteht aus einer geordneten Liste von Darstellungseinheiten (*Displayunit*), der weitere Untereinheiten zugeordnet sein können (*displayunits*). Eine Darstellungseinheit beinhaltet eine geordnete Liste (*displayelements*) von Darstellungselementen (*Displayelements*). Ein Darstellungselement kann entweder eine Gruppe von Darstellungselementen (*Fieldgroup*) oder ein Interaktionselement für ein Eingabefeld sein (*Field*).

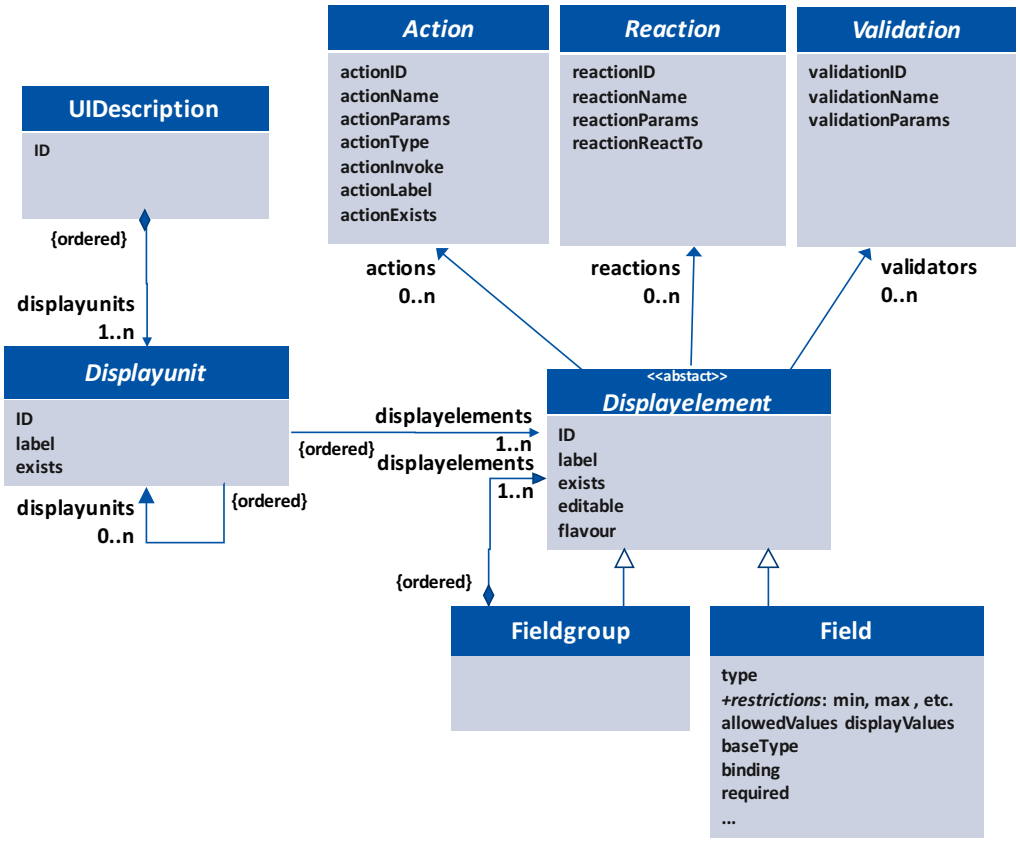


Abbildung 8.4. Modell zur Beschreibung dialogbasierter Benutzungsschnittstellen

Interaktionselemente zur Ein-/Ausgabe. Die Beschreibung der Interaktionselemente erfolgt über den Typ des zu erfassenden Datums. Zudem können Restriktionen angegeben werden, welche die Eingabe näher beschreiben (vgl. Abschnitt 5.4).

Für das Benutzungsschnittstellenmodell wird eine Menge an Basistypen und Restriktionen festgelegt, die bei der Modellierung verwendet werden. In Anhang A.2.3, Tabelle A.10 sind diese dargestellt. Die Beschreibung der Benutzungsschnittstelle erfolgt ausschließlich über diese Basistypen. Damit ist sichergestellt, dass die Benutzungsschnittstelle aus einem definierten Satz an Elementen aufgebaut ist. Dies gestattet später die automatische Erzeugung einer Benutzungsschnittstelle durch einen Generator, der die Basistypen auf Standard-Interaktionselemente abbilden kann.

Sollen bei der Erzeugung spezifische Interaktionselemente verwendet werden, können mögliche Varianten des Basistyps als *Geschmacksrichtung* (*flavour*) des Elements angegeben werden. So könnte z.B. für eine M-aus-N-Auswahl eine Variante *circularSelection* angegeben werden, welche zur Darstellung des Körperatlas (Abschnitt 5.4) verwendet werden kann. Der Generator erhält damit weitere Hinweise zur Auswahl eines Darstellungselements und kann gezielt spezifische Interaktionselemente generieren.

Verhalten von Darstellungselementen. Für Darstellungselemente wird das Verhalten analog dem Vorgehen im Anwendungsmodell über Aktionen (*Action*), Reaktionen (*Reaction*), Validatoren (*Validator*) und Modellbedingungen (*ModelCondition*) modelliert werden (vgl. Abschnitte 5.5 und 6.3.4). Für alle Darstellungselemente können Regeln für deren

Sicht-/Editierbarkeit (*exists* und *editable*), Validierungen (*validations*), Aktionen (*actions*) Reaktionen (*reactions*) angegeben werden (vgl. Abb. 8.4). Die Attribute zur Beschreibung der Operationen entsprechen dabei den Angaben, die auch im Anwendungsmodell aus Abschnitt 6.3.4 enthalten sind. Die Attribute sind in Anhang A.2.3 zusammengefasst.

8.4.2 Herleitung der abstrakten Benutzungsschnittstelle (AUI)

Die AUI-Transformation erzeugt aus dem Anwendungsmodell eine erste Version des Benutzungsschnittstellenmodells (AUI-Modell), in welchem die Datenelemente auf Interaktionselemente des Benutzungsschnittstellenmodells abgebildet werden. Diese werden hier noch abstrakt und unabhängig von einem Ausgabemedium spezifiziert (vgl. CAMELEON-Framework [25]). Die Struktur des Anwendungsmodells bleibt bei dieser Transformation erhalten. Die Gruppierung von Elementen und hierarchische Beziehungen werden aus dem Strukturbaum des Anwendungsmodells übernommen. Die Grundlage zur Auswahl der Interaktionselemente bilden das **Anwendungsmodell und der Nutzungskontext** der Anwendung.

Das **Anwendungsmodell** liefert grundlegende Informationen zum Datenelement (Typ, geltende Restriktionen, semantische Annotationen) zur Bestimmung des zu nutzenden Interaktionselements. Der **Nutzungskontext** bestimmt darüber hinaus ggf. alternative Interaktionselemente, die sich für ein spezifisches Umfeld ergeben. Dies ist z.B. der Einsatz der Anwendung in unterschiedlichen Ländern und daraus resultierenden alternativen Eingabeformen. Der Nutzungskontext enthält dabei ergänzende Regeln, die die Auswahl eines Interaktionselements beeinflussen.

Vorgehensweise zur AUI-Transformation

Algorithmus 3 zeigt das Vorgehen zur AUI-Transformation als Pseudocode. Als initialer Eingabeparameter für die Transformation dient der Wurzelknoten des Strukturbaums des Anwendungsmodells, dessen Knoten über eine *Preorder-Traversierung* [42] besucht werden, welche die Reihenfolge der Unterknoten erhält.

Die Funktion *transformAMTreeToAUITree* transformiert einen Teilbaum des Anwendungsmodells in einen korrespondieren Teilbaum des AUI-Modells. Für jeden Knoten des Strukturbaums wird dabei ein korrespondierendes Darstellungselement erzeugt (*createDisplayelementFor (amNode)*). Die Operation wird rekursiv auf die untergeordneten Teilbäume ausgeführt und die Ergebnisse dem aktuellen Displayelement zugewiesen.

Die zur Erzeugung der AUI-Knoten dargestellte Funktion (*createDisplayelementFor*) generiert AUI-Knoten aufgrund der im Strukturbaumknoten enthaltenen Attribute und des Nutzungskontexts. Für Datenelemente wird hierbei zuerst betrachtet, ob der Nutzungskontext die Verwendung eines alternativen Interaktionselements erfordert. Ist dies nicht der Fall, wird basierend auf den Typinformationen des Ausgangsknotens ein Interaktionselement bestimmt. Im Anwendungsmodell enthaltene Informationen zu Restriktionen, der Semantik und dem Verhalten werden für die spätere Verwendung übertragen.

Das Resultat ist ein AUI-Strukturbaum, der einen zum Strukturbaum der Anwendung identischen Aufbau aufweist. Die Knoten des Baums beschreiben Darstellungselemente des Be-

Algorithmus 3 AUI-Transformation

```
1:  function transformAMTreeToAUITree(amNode)
2:      environment
3:          contextOfUse (Nutzungskontext)
4:      inputs
5:          amNode (Knoten im AM)
6:      result displayelement (Knoten im AUI)
7:      begin
8:          displayelement ← createDisplayelementFor (amNode, contextOfUse)
9:          for amNode.children do
10:             subtree ← transformAMTreeToAUITree (child)
11:             displayelement.appendChild (subtree)
12:     return displayelement

13: function createDisplayelementFor(amNode, contextOfUse)
14:         if amNode ofType innerNode then
15:             newDisplayelement ← new Group ()
16:             newDisplayelement.mapGroupInformationFrom (amNode, contextOfUse)
17:         if amNode ofType leafnode then
18:             newDisplayelement ← new Field ()
19:             newDisplayelement.mapElementInformationFrom (amNode, contextOfUse)
20:     return newDisplayelement
```

nutzungsschnittstellenmodells, die aufgrund der Typinformationen im Anwendungsmodell und den Informationen des Nutzungskontexts bestimmt wurden.

Beispiel für die Transformation

Listing 8.1 zeigt das Anwendungsmodell für die Erfassung von *E-Mail*, *Postleitzahl* und *Ort* in DSL-Syntax. *E-Mail* und *Postleitzahl* werden durch semantische Annotationen (*ext:email*, *ext:zipcode*) näher beschrieben, die bei der Transformation berücksichtigt werden. *Ort* enthält die Beschreibung einer Reaktion auf die Änderung des Inhalts des Feldes *zipcode*.

```
email : {
    semanticTag="ext:eMail"
}
zipcode : {
    semanticTag="ext:zipcode"
}
city : {
    reactions="zipcode:ext_liability.prefillCity(<pathTo>.zipcode,$<...
        ↪ pathTo>.city)"
}
```

Listing 8.1: Datenelemente des Anwendungsmodells

In Listing 8.2 ist das Transformationsergebnis im Benutzungsschnittstellenmodell für den Länderkontext *Deutschland* dargestellt. Aufgrund der semantischen Annotation *ext:email* bestimmt der Transformator eine Darstellung der E-Mail als Texteingabe und eine syntaktische Validierung des Feldes über einen regulären Ausdruck für E-Mail-Adressen.

Für das Feld *Postleitzahl* (*zipcode*) wurde eine länderspezifische Variante generiert. Im deutschen Kontext wird das Datenelement auf eine Zahleneingabe (*number*) abgebildet, dessen Eingabe auf fünf Ziffern beschränkt ist. Zur Eingabe des Ortes (*city*) wird ein Textfeld verwendet, welches mit einer Reaktionsoperation versehen ist (*reactions*). Hierbei wurden die im Anwendungsmodell angegebenen Informationen zum Anstoßen einer Reaktionsoperation in das Benutzungsschnittstellenmodell übertragen.

```
      {
        "label":"E-Mail",
        "id":"<pathTo>.eMail",
        "type":"text",
        "regEx":"[a-zA-Z._]{0,}@[a-zA-Z._-]*$"
      },
      {
          "label":"Postleitzahl",
          "id":"<pathTo>.zipcode",
          "type":"number",
          "length":"5"
      },
      {
          "label":"Ort",
          "id":"<pathTo>.city",
          "type":"text",
          "reactions":[{
            "reactionId"      :"ext_liability.prefillCity_id",
            "reactionName"    :"ext_liability.prefillCity",
            "reactionParams"  :"<pathTo>.zip,$<pathTo>.city",
            "reactionReactTo" :"<pathTo>.zip"
            }]
      } ... ]
```

Listing 8.2: Interaktionselement für E-Mail-Adresse

Listing 8.3 zeigt die Unterschiede im Transformationsergebnis für das Feld *Postleitzahl* für den Länderkontext *Großbritannien*. Hier bestimmt der Nutzungskontext eine alternative Auswahlregel für das Element. Es wird ein Textfeld (*text*) verwendet, das eine Operation für die länderspezifische Validierung von Postleitzahlen[43] aufruft (*ext_fn:validateZIP_UK*).

```
  {
      "label":"Zip Code",
      "id":"<pathTo>.zipcode",
      "type":"text",
      "maxlength":"10",
      "validators":[ {
        "validatorName":"ext_fn.validateZIP_UK",
        "validatorParams":"<pathTo>.zip"
      }]
  }
```

Listing 8.3: Kontextabhängige Interaktionselemente für Postleitzahl

[43] vgl. https://de.wikipedia.org/wiki/Postleitzahl_(Vereinigtes_Königreich)

8.4.3 Herleitung der konkreten Benutzungsschnittstelle (CUI)

Die CUI-Transformation erzeugt aus dem AUI-Modell ein Benutzungsschnittstellenmodell für einen spezifischen Interaktionskontext. Der Strukturbaum des AUI-Modells wird dabei umstrukturiert, sodass er für den Interaktionskontext sinnvolle Darstellungseinheiten beschreibt. Dies umfasst insbesondere die

- Bildung von Darstellungseinheiten und ggf. untergeordneten Einheiten
- Verteilung der Darstellungselemente auf Darstellungseinheiten

Im Falle grafischer Benutzungsschnittstellen bedeutet dies z.B. konkret die Verteilung der Fragen auf Seiten und Unterseiten.

Der Interaktionskontext bestimmt die **Strategie**, mit welcher der AUI-Strukturbaum modifiziert wird. Die zur Durchführung der Strategie relevanten Informationen stammen einerseits aus dem Interaktionskontext, andererseits aus dem Anwendungsmodell. Der **Interaktionskontext** bestimmt die Rahmenbedingungen für die Transformation. Diese legen z.B. fest, ob Untereinheiten verwendet werden können, wie viele Elemente minimal/maximal je Einheit sinnvoll sind und ob Gruppen in eigene Einheiten ausgelagert werden können.

Das **Anwendungsmodell** enthält die benötigten Informationen zur Struktur, Gruppierung, Kohäsion und der Sequenz der Elemente. Hieraus kann abgeleitet werden,

- welche Elemente gemeinsam auf Unterseiten ausgelagert werden können
- an welchen Stellen der Fragefluss unterbrochen und auf einer Folgeseite fortgesetzt werden kann
- welche Gruppen auf Unterseiten ausgelagert werden können

Die Strategie bei der CUI-Transformation variiert abhängig vom Interaktionskontext. So könnte beispielsweise eine Strategie für *mobile Geräte* vorsehen, dass alle Fragen auf einer Seite stehen sollen und damit auch keine Gruppen auf Unterseiten ausgelagert werden sollen. Eine Strategie für *desktop computer* hingegen könnte diese Einschränkungen aufheben, was in einer strukturell komplexen Benutzungsschnittstelle resultiert.

Vorgehensweise zur CUI-Transformation

In Algorithmus 4 ist der Ablauf der CUI-Transformation dargestellt. Als Eingabe erhält der Algorithmus den zu modifizierenden AUI-Strukturbaum und den Interaktionskontext. Zu Beginn wird eine zum Interaktionskontext passende Strategie zur Erzeugung von Darstellungseinheiten und der Verteilung der Darstellungselemente gewählt. Die gewählte Strategie wird auf den AUI-Strukturbaum angewendet (*strategy.paginateTree()*). Dabei werden dem AUI-Strukturbaum Darstellungseinheiten hinzugefügt, ggf. Untereinheiten erzeugt und die im AUI enthaltene Darstellungselemente auf diese Einheiten verteilt. Florins et al. [67] nennen Strategien zur Seitenaufteilung, die auf der AUI- bzw. CUI-Ebene erfolgen können (vgl. auch [47, 207]).

Das Resultat der Transformation ist eine Strukturbaumvariante des AUI-Modells, dessen Darstellungselemente auf Darstellungseinheiten verteilt sind. Sofern dies möglich ist, wurden untergeordnete Darstellungseinheiten gebildet und die Navigation angepasst.

Algorithmus 4 CUI-Transformation

```
1:  function transformAUITreeToCUITree
2:      environment
3:          interactionContext (Interaktionskontext)
4:      inputs
5:          auiTree (AUI-Strukturbaum)
6:      result cuiTree (CUI-Strukturbaum)
7:      begin
8:          strategy ← selectPaginationStrategyFor (interactionContext)
9:          paginatedTree ← strategy.paginateTree (auiTree, interactionContext)
10:     return paginatedTree
```

Beispiel für die Transformation

In Listing 8.4 ist ein vereinfachtes Anwendungsmodell der Gruppen zur Erfassung von Kundendaten dargestellt (ohne Datenelemente). Es zeigt die Gruppierung der Informationen zur Erfassung des Namens (*fullname*), der Geburtsdaten (*birthdata*), des Familienstands (*maritalinformation*) und der Adresse (*address*). Die Gruppe *address* weist dabei eine schwache Kohäsion zum Umfeld auf (*cohesion=weak*).

```
data_description: "liability" {
    group : "customerdata"  {
        group : "fullname"  { ... }
        group : "birthdata" { ... }
        group : "maritalinformation" { ... }
        group : "address"  {cohesion="weak"} { ... }
}
```

Listing 8.4: Anwendungsmodell (Gruppen) zur Erfassung von Kundendaten

Das Ergebnis der Transformation für den Interaktionskontext für Desktop-Anwendungen ist in Listing 8.5 dargestellt. Die Strategie für diesen Kontext sieht vor, dass für alle Gruppen auf oberster Ebene der Beschreibung eine Darstellungseinheit angelegt wird. Gruppen bleiben in ihrer Schachtelungstiefe erhalten. Eine Ausnahme bilden Gruppen, die mit schwacher Kohäsion gekennzeichnet sind. Diese sollen auf eine neue Darstellungseinheit ausgelagert werden und der nächst höheren Einheit der Hierarchie untergeordnet werden.

Für die Kundendaten wurde dem Modell eine Seite hinzugefügt ①, die auf oberster Ebene die Kundendaten zusammenfasst. Die in den Kundendaten enthaltenen Gruppen des Anwendungsmodells werden auf Gruppen des Benutzungsschnittstellenmodells abgebildet und bleiben in ihrer Struktur erhalten ②③④. Eine Ausnahme bildet die Gruppe *address*, welche im Anwendungsmodell mit schwacher Kohäsion angegeben ist. Für diese Gruppe wurde eine Unterseite der Kundendaten angelegt ⑤, welche die Gruppe *address* enthält ⑥.

Die Strategie für mobile Anwendungen sieht hingegen vor, dass alle Elemente auf einer Darstellungseinheit untergebracht werden. Das Ergebnis der Transformation für diesen Fall ist in Listing 8.6 dargestellt. Das Benutzungsschnittstellenmodell enthält in diesem Fall genau eine Darstellungseinheit für die Anwendung ①. Die im Anwendungsmodell enthaltenen Gruppen werden nacheinander auf einer Hierarchiestufe dargestellt ②. Die Gruppe *address* verbleibt im Fragefluss ③. Die Darstellungseinheit besitzt damit keine Untereinheiten ④.

```
   {
     "applicationName":"liability",
     "displayunits":[
❶       { "_nodetype" :"displayunit",
          "id"        :"liability.customerdata_page",
          "displayelements": [
❷           { "_nodetype"   :"group",
              "id"          :"liability.customerdata.fullname",
              "name"        :"fullname",
              "displayelements":[ ... ]
             },
❸           { "_nodetype"   :"group",
              "id"          :"liability.customerdata.birthdata",
              "displayelements":[    ... ]
             },
❹           { "_nodetype"  :"group",
              "id"         :"liability.customerdata.maritalinformation",
              "displayelements":[ ... ]
             }
            ],
❺         "displayunits":
            [
             { "_nodetype"   :"displayunit",
              "id"          :"liability.customerdata.address_page",
              "displayelements":[
❻               { "_nodetype"  :"group",
                  "id"         :"liability.customerdata.address",
                  "displayelements":[ ... ]
                 }
            ...
```

Listing 8.5: Ergebnis der CUI-Transformation für Desktop-Anwendungen

```
    {
     "applicationName":"liability",
     "displayunits":[
❶       { "_nodetype" :"displayunit",
          "id"        :"liability_page",
          "displayelements": [
❷           { "_nodetype"   :"group",
              "id"          :"liability.customerdata.fullname",
              "name"        :"fullname",
              "displayelements":[ ... ]
             },
              ...
❸           { "_nodetype"  :"group",
              "id"         :"liability.customerdata.address",
              "displayelements":[ ... ]
             }
            ],
❹        "displayunits":
            []
            ...
```

Listing 8.6: Ergebnis der CUI-Transformation für mobile Anwendungen

8.4.4 Herleitung der finalen Benutzungsschnittstelle (FUI)

Die FUI-Transformation erzeugt aus dem erzeugten CUI-Modell eine Benutzungsschnittstelle für eine bestimmte Plattform in einem spezifischen **Technologiekontext**. In diesem Schritt werden für die noch abstrakt spezifizierten Elemente des Benutzungsschnittstellenmodells konkrete Interaktionselemente in der Technologie erzeugt, die auf der Zielplattform zur Verfügung steht.

Das konkrete Vorgehen und das Ergebnis des Transformationsschritts hängt in hohem Maße von der Zieltechnologie und Plattform ab. Hierbei muss unterschieden werden, ob eine Zieltechnologie eine **deklarative Beschreibung der Benutzungsschnittstelle** verarbeiten kann oder ob **Code zu deren Erzeugung** ausgeführt werden muss. Zudem erfordern einige Technologien, dass die Benutzungsschnittstellen bereits zum Zeitpunkt der Entwicklung feststehen, andere können UIs dynamisch zur Laufzeit Benutzungsschnittstellen erzeugen.

Für eine Vielzahl von Technologien existieren deklarative Methoden zur Beschreibung von Benutzungsschnittstellen: z.B. *Hypertext Markup Language (HTML)*[63], *XML User Interface Language (XUL)*[149], *XForms*[53] oder *VoiceXML*[137] sind Beschreibungssprachen, aus denen zur Laufzeit Benutzungsschnittstellen erzeugt werden. Aber auch gerätespezifische Technologien besitzen häufig Möglichkeiten zur deklarativen Beschreibung: so verwenden iOS und macOS *Xcode Interface Builder (XIB)*-Dateien, die in eine Benutzungsschnittstelle übersetzt werden. In Microsoft Windows wird analog die *Extensible Application Markup Language (XAML)*[44] angeboten. *JavaFX* sieht die Verwendung von *JavaFX Markup Language (FXML)*-Dateien[45] vor. Für Android-Anwendungen werden *Layout Resource*-Dateien verwendet[46]. Für einige Zielplattformen existieren hingegen keine deklarativen Ansätze. Hier müssen Benutzungsschnittstellen bereits während der Entwicklung im Code der Anwendung spezifiziert oder zur Laufzeit dynamisch erzeugt werden (z.B. JavaSwing, JavaAWT).

Das Ergebnis der Transformation für deklarative Technologien ist eine **plattformspezifische Benutzungsschnittstellenbeschreibung** in der vom Technologiekontext vorgegebenen Beschreibungssprache (z.B. *XIB-, XAML-, FXML-* oder *Layout Resource*-Datei). Diese Beschreibung kann entweder während der Entwicklung oder zur Laufzeit zur Erzeugung der Benutzungsschnittstelle verwendet werden.

Für Technologien, die keinen deklarativen Ansatz verfolgen, bestehen zwei Optionen: Hier könnte entweder **konkreter Code generiert** werden oder die **FUI-Transformation zur Laufzeit in der Anwendung** erfolgen. Die erste Option ermöglicht lediglich die Verwendung der Transformationsergebnisse zum Zeitpunkt der Entwicklung. Die Verlagerung der FUI-Transformation in die Anwendung hingegen ermöglicht eine dynamische Erzeugung der Benutzungsschnittstelle zur Laufzeit.

Im Folgenden wird zuerst die generelle Vorgehensweise für eine FUI-Transformation aufgezeigt. Die Beschreibung ist unabhängig davon, ob das Ergebnis eine deklarative Beschreibung ist oder ob die Erzeugung der Elemente zur Laufzeit erfolgt. Die weiteren Darstellungen konzentrieren sich auf deklarative Ansätze. Das Verfahren wird für diesen Fall konkretisiert

[44] https://msdn.microsoft.com/de-de/library/cc295302.aspx

[45] https://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm

[46] https://developer.android.com/guide/topics/resources/layout-resource.html

und das Transformationsergebnis exemplarisch für HTML-basierte Benutzungsschnittstellen dargestellt.

Allgemeine Vorgehensweise zur FUI-Transformation

Die Transformation in die finale Benutzungsschnittstelle ist eine Operation auf dem Strukturbaum des CUI-Modells (Abschnitt 8.4.3). Die folgenden Aufgaben sind von der FUI-Transformation zu erfüllen:

- Erzeugung **konkreter Darstellungseinheiten, Gruppierungs- und Interaktionselemente** in der vom CUI-Modell vorgegebenen Struktur
- Auswahl **konkreter Interaktionselemente** für die Ein-/Ausgabe
- Nutzung der **Beziehungen zum Anwendungskontext** zur Umsetzung der Datenhaltung und des Verhaltens

Hierzu wird der CUI-Strukturbaum durchlaufen und dabei für jeden Knoten ein entsprechendes FUI-Element ausgewählt, das zu Typ und Semantik des CUI-Elements passt und mit zusätzlichen Materialien aus dem Nutzungskontext angereichert. Daraus wird eine entsprechende Repräsentation in der Zieltechnologie erzeugt (z.B. die deklarative Beschreibung des Elements oder dessen konkrete Instantiierung zur Laufzeit).

Die zur Transformation benötigten Informationen stammen einerseits aus dem CUI-Modell, andererseits aus dem Technologie- und dem Nutzungskontext der Anwendung. Das **CUI-Modell bestimmt** den Aufbau der Benutzungsschnittstelle aus einem definierten Satz an Standard-Interaktionselementen. Zusätzlich enthält es ggf. Informationen zur Semantik der Elemente. Der **Technologiekontext bestimmt**, wie die CUI-Elemente in konkrete FUI-Elemente transformiert werden und ob für Elemente mit einer bestimmten Semantik applikationsspezifische FUI-Elemente erzeugt werden können. Der **Nutzungskontext liefert** zu jedem Element zusätzliche Materialien (*Assets*), die für die finale Darstellung der Benutzungsschnittstelle benötigt werden. Dies sind z.B. Texte, Beschriftungen und Abbildungen (vgl. Abschnitt 5.6).

FUI-Transformation für deklarative Beschreibungssprachen

Algorithmus 5 stellt die Vorgehensweise zur Erzeugung einer deklarativen Benutzungsschnittstellenbeschreibung dar. Hierzu werden folgende Annahmen getroffen. (1) Es existiert ein Generator, der basierend auf Vorlagen (*Templates*) textuelle Repräsentationen für FUI-Elemente erzeugt. (2) Es existieren Vorlagen für jedes Standard-Interaktionselement und ggf. weitere für applikationsspezifische Elemente. Diese Vorlagen sind im Technologiekontext hinterlegt. (3) Der Nutzungskontext enthält die zur Erzeugung der finalen Schnittstelle benötigten Materialien.

Als Eingangsparameter erhält der Algorithmus (*transformCUITreeToFUI()*) den Wurzelknoten des CUI-Strukturbaums, dessen Knoten über eine *Preorder-Traversierung* [42] besucht werden. Im ersten Schritt werden für das übergebene CUI-Element die zum Nutzungskontext passenden Materialen erfragt. Im nächsten Schritt wird Template bestimmt, das zum Technologiekontext, zum Typ und den semantischen Tags des Elements passt (*getTemplate()*). Dieses wird zur Generierung der deklarativen Repräsentation unter Angabe der Materialien verwendet (*generateCode()*). Abschließend wird die Operation rekursiv auf alle Unterknoten

des CUI-Elements angewendet. Das Ergebnis ist eine deklarative Beschreibung der Benutzungsschnittstelle, in welcher für jeden Knoten des CUI-Strukturbaums eine entsprechende Repräsentation erzeugt wurde.

Algorithmus 5 FUI-Transformation in eine textuelle Beschreibung

```
1:  procedure TRANSFORMCUITREETOFUI(cuiTreeNode)
2:      environment
3:          contextOfUse (Nutzungskontext)
4:          technologyContext (Technologiekontext)
5:      inputs
6:          cuiTreeNode (CUI-Strukturbaumknoten)
7:      begin
8:          assets ← contextOfUse.getAssetsFor(cuiTreeNode.id)(interactionContext)
9:          template ← technologyContext.getTemplate(cuiTreenode.type, cuiTree-
    node.semanticTags);
10:         generator.generateCode (template, cuiTreeNode, assets);
11:         for cuiTreeNode.children do
12:             transformCUITreetoFUI (child)
```

Beispiel für Transformationsergebnisse (HTML)

Im Folgenden werden exemplarisch Transformationsergebnisse für Felder und Gruppen dargestellt. Die Darstellung erfolgt am Beispiel von HTML als Zieltechnologie. Listing 8.7 zeigt CUI-Repräsentationen für die Felder zur Angabe der Postleitzahl ① und des Familienstands ②. Als Beispiel für eine Gruppe sind die Partnerdaten dargestellt ③. Das CUI-Modell enthält die Typinformationen für die Felder (*type*). Für die Postleitzahl ist zudem eine Validierung angegeben (*validators*). Für die Gruppe ist eine Existenzbedingung angegeben, die dessen Anzeige abhängig von der Auswahl des Familienstandes beschreibt (*exists*).

```
❶ { "_nodetype":"field",
      "id":"<pathTo>.zip",
      "type":"number",
      "validators":[{
         "validatorName":"fn_ext.validateZIP",
         "validatorParams":"<pathTo>.zip"}]
      }
❷ { "_nodetype":"field",
      "id":"<pathTo>.status",
      "required":"",
      "type":"oneOfMany",
      "allowedValues":"marriednotmarrieddivorced"
      "value":"married",
    }
❸  { "_nodetype":"group",
      "id":"<pathTo>.partnerinformation",
      "exists":"(<pathTo>.status == 'married')",
      "displayelements":[ ... ]
    }
```

Listing 8.7: Gruppen- und Feldelemente des CUI-Modells

Listing 8.8-8.10 stellen den HTML/JavaScript-Code dar, der durch eine FUI-Transformation für eine webbasierte Benutzungsschnittstelle erzeugt wird. Die hier dargestellte Variante ist für einen deutschen Sprachkontext bestimmt und verwendet u.a. AngularJS [82] als Framework zur Umsetzung der funktionalen Aspekte der Benutzungsschnittstelle.

```
   <div id="<pathTo>.zip" class="form-group row" >
❶   <label class="control-label ...">Postleitzahl</label>
       <div class="form-control-wrapper ... ">
❷         <input  class="form-control ... "
                  type="number"
                  ng-model="<pathTo>.zip"
❸                mimesis-validate="ext_fn.validateZIP(<pathTo>.zip)"
                  ng-model-options="{ updateOn: 'blur' }" update-OnEnter
             />
       </div>
    </div>
```

Listing 8.8: Erzeugter HTML-Code für das Feld Postleitzahl

Für die Postleitzahl wird das in Listing 8.8 dargestellte HTML-Code-Fragment erzeugt, welches mit einer deutschsprachigen Beschriftung versehen wird ①. Das Eingabeelement ist ein Zahlenfeld ②, für welches eine Validierungsoperation gemäß dem CUI-Element angegeben ist, die bei Verlassen des Feldes ausgeführt wird ③.

Der Bezug zum Anwendungskontext wird über Attribute der HTML-Elemente hergestellt. In der hier exemplarisch dargestellten Implementierung für AngularJS werden *Direktiven*[47] verwendet, die bei der späteren Ausführung ausgewertet werden. Sie beschreiben, wo die Eingaben abgelegt werden (*ng-model*) bzw. welche Operationen ausgeführt werden sollen (z.B. *mimesis-validate* zur Validierung der Eingabe).

```
   <div id="<pathTo>.status" class="form-group row" >
❶     <label class="control-label ... "/>Familienstand *</label>
       <div class="form-control-wrapper ... ">
❷        <select class="form-control"
                 id="<pathTo>.status"
                 ng-required="required"
                 ng-model="<pathTo>.status">
                   <option value="married">verheiratet</option>
                   <option value="notmarried">ledig</option>
                   <option value="divorced">geschieden</option>
           </select>
       </div>
   </div>
```

Listing 8.9: Erzeugter HTML-Code für das Feld Familienstand

Für das Feld *Familienstand* wird aufgrund der Typinformation im CUI-Element (*type:oneOfMany*) eine Auswahlliste erzeugt, welche die in *allowedValues* angegebenen Werte zur Auswahl anbietet (Listing 8.9, ②). Die Beschriftung ① enthält die deutsche Bezeichnung des Feldes sowie eine Markierung, die anzeigt, dass es sich um ein Pflichtfeld handelt.

[47] vgl. z.B. https://angularjs.de/buecher/angularjs-buch/angularjs-directives/

Listing 8.10 zeigt das erzeugte HTML-Code-Fragment für die Gruppe der Partnerinformationen. Diese wird als HTML–Bereich ① mit einem deutschsprachigen Titel erzeugt ②. Für den Bereich ist zudem die Existenzbedingung angegeben, welche die Sichtbarkeit des Elements bestimmt ③.

```
❶  <div bs-panel
❷        title="Partner Informationen"
       id="<pathTo>.partnerinformation"
❸     mimesis-exists="(<pathTo>.status == 'married')"
       mimesis-modelid="<pathTo>.partnerinformation"
     >
    ...
    </div>
```

Listing 8.10: Erzeugter HTML-Code für die Gruppe Partnerinformationen

8.5 Darstellung und Ausführung (Rendering in der Laufzeitumgebung)

Die Zielplattform stellt die Laufzeitumgebung der Benutzungsschnittstelle zur Verfügung. Sie erhält die finale Benutzungsschnittstellenbeschreibung als Eingabe und erzeugt daraus eine lauffähige Instanz der Anwendung. In Abschnitt Abschnitt 5.2 wurde der grundsätzliche Aufbau einer Laufzeitumgebung dargestellt.

Die Aufgaben der Laufzeitumgebung bestehen dabei einerseits in der Erzeugung der Präsentation (Erzeugung der Darstellungseinheiten, Interaktionselemente, Navigation) andererseits in der Steuerung der Anwendung zur Laufzeit (Navigation, Verwaltung der erfassten Daten, Ausführung von Operationen). Hierzu stellt die Umgebung einen Anwendungskontext bereit, der die Daten und Operationen der Anwendung verwaltet.

Am Beispiel einer HTML-basierten Oberfläche ist der HTML-Browser die Laufzeitumgebung. Dieser interpretiert die HTML-Beschreibung und erzeugt einen DOM-Baum, welcher die Struktur- und Eingabeelemente repräsentiert und darstellt. Zur Umsetzung des dynamischen Verhaltens werden die DOM-Elemente mit Aktionsauslösern (*Trigger*) versehen, die JavaScript-Operationen aufrufen.

Zur Laufzeit werden die Eingaben des Nutzers in den Anwendungskontext geschrieben. Zur Umsetzung des Verhaltens benötigte Operationsaufrufe werden dem Anwendungskontext zur Bearbeitung übergeben, der die konkrete Implementierung zur Verfügung stellt.

8.6 Diskussion der Ergebnisse

Das Ziel dieses Kapitels lag in der Spezifikation eines Verfahrens, welches aus einem datenzentrierten Anwendungsmodell und einer Variantenbeschreibung konkrete, ausführbare Benutzungsschnittstellenvarianten erzeugt. Hierzu wurde ein Prozess beschrieben, der in drei Phasen (*Variantentransformation, Benutzungsschnittstellentransformation, Darstellung und Ausführung in der Laufzeitumgebung*) schrittweise aus den zugrunde gelegten Modellen eine finale Benutzungsschnittstelle generiert und zur Anzeige bringt. Das vorgestellte Verfahren zeigt, dass es möglich ist, aus einem technologieneutralen, datenzentrierten Modell vollständig automatisiert eine konkrete Benutzungsschnittstelle herzuleiten.

Zur Verifikation der Erfüllung der Anforderungen sind in Tabelle 8.5 die Ziele der einzelnen Phasen und enthaltenen Transformationsschritte zusammengefasst. Hier ist dargestellt, welche Anforderungen in den einzelnen Schritten umgesetzt werden.

Phase / Transformation	Ziel der Transformation	Umsetzung Anforderungen
Variantentransformation	Erzeugung eines **variantenspezifischen Modells** zur Erzeugung der Benutzungsschnittstelle	**(A3.1)**
Durchführung Komposition	Erstellung eines **vollständigen Anwendungsmodells** durch Inklusion und Anpassung referenzierter Komponenten	
Erstellung Variantenmodell	Erzeugung einer **inhaltlichen Variante** aufgrund der Variantenbeschreibung	**(IR.4) (IR.9) (IR.15) (IR.16)**
Benutzungsschnittstellentransformation	Erzeugung einer finalen, **ausführbaren** Benutzungsschnittstellenbeschreibung	**(A3.2)**
abstrakte Benutzungsschnittstelle (AUI)	Überführung des datenzentrierten Anwendungsmodells in ein akstraktes Benutzungsschnittstellenmodell für einen konkreten **Nutzungskontext**	**(IR.5) (IR.6) (IR.7) (IR.10) (IR.11) (IR.12) (IR.13) (IR.14)**
konkrete Benutzungsschnittstelle (CUI)	Erzeugung eines Benutzungsschnittstellenmodells für einen konkreten **Interaktionskontext**	**(IR.1) (IR.2) (IR.3)**
finale Benutzungsschnittstelle (FUI)	Erzeugung einer Benutzungsschnittstelle für einen konkreten **Technologiekontext**	**(IR.6) (IR.7) (IR.8) (IR.17)**
Darstellung und Ausführung	Rendering der Benutzungsschnittstelle auf der konkreten **Zielplattform** und Bereitstellung des **Anwendungskontexts**	**(A3.3)**

Abbildung 8.5. Zuordnung der Anforderungen zu Phasen und Prozessschritten

Umsetzung der Anforderungen an die Generierung. Das Verfahren setzt die Anforderungen (A3.1)-(A3.3) um. In Phase *Variantentransformation* wird aus dem Anwendungsmodell und der Variantenbeschreibung ein Variantenmodell hergeleitet, welches sowohl referenzierte **Komponenten integriert** als auch **inhaltliche Varianten** erzeugt, die aus der Variantenbeschreibung automatisch abgeleitet werden (A3.1). In der Phase *Benutzungsschnittstellentransformation* wird schrittweise eine **finale technische Variante der Benutzungsschnittstelle** des Variantenmodells erzeugt, die auf einen konkreten Nutzungs-, Interaktions- und Technologiekontext zugeschnitten ist (A3.2). Die daraus resultierende Schnittstelle wird von einer Laufzeitumgebung zur Darstellung gebracht, die den Anwendungskontext zur Verfügung stellt und damit die Benutzungsschnittstelle ausführen kann (A3.3).

Umsetzung der Eigenschaften dialogbasierter Benutzungsschnittstellen. Die grundlegende Anforderung an das Verfahren ist, dass Benutzungsschnittstellenvarianten generiert werden, welche die in der Analyse (Kapitel 5) identifizierten Eigenschaften aufweisen. Hierzu muss es die Anforderungen (IR.1)-(IR.17) aus den Tabellen 5.1-5.4 (vgl. Abschnitt 5) bei der Generierung vollständig umsetzen. Das dargestellte Verfahren berücksichtigt diese An-

forderungen an unterschiedlichen Stellen des Prozesses. In Tabelle 8.5 ist aufgeführt, welche der Anforderungen (IR.1)-(IR.17) in welchem Transformationsschritt des Generierungsprozesses umgesetzt werden.

Offene Fragestellungen

Obwohl im Verlauf der Arbeit das dargestellte Verfahren sinnvolle Ergebnisse für den betrachteten Problemraum liefert, sind dennoch Fragen offen, die nicht abschließend untersucht wurden.

Konkretisierung referenzierter Kontexte: Im dargestellten Verfahren wurde nicht spezifiziert, wie der Nutzungs-, Interaktions- und Technologiekontext beschrieben werden kann und welche Informationen darin enthalten sein müssen. Es wurden lediglich exemplarisch Informationen genannt, die aus dem jeweiligen Kontext bezogen werden (z.B. Sprach- und Länderbezug im Nutzungskontext, Vorlagen zur Generierung im Technologiekontext). Es wurde nicht spezifiziert, wie diese Informationen repräsentiert werden oder deren Vollständigkeit nachgewiesen.

Im Rahmen der Evaluation der Arbeit wurden exemplarisch grundlegende Informationen verwendet (z.B. Labels, Hilfetexte, feste Algorithmen zur Seitenaufteilung), jedoch festgestellt, dass ggf. weitere Aspekte beachtet werden müssen, damit das Verfahren für beliebige Kontexte nutzbar wird (z.B. weitere Arten von Materialien zur Erzeugung sprachbasierter Anwendungen). Um universell einsetzbar zu sein, müssen umfänglich Inhalte identifiziert und eine Spezifikationsform gefunden werden, die potentiell auf alle Kontexte anwendbar ist. Grundlagen hierfür finden sich z.B. in Arbeiten von Peißner et al. [166, 167], auf denen zukünftige Arbeiten aufsetzen können.

Universalität des Verfahrens: In Abschnitt 8.4.4 beschränkte sich die Darstellung auf deklarative FUI-Beschreibungen als Ergebnis der FUI-Transformation. In der beschriebenen Form eignet sich das Verfahren für Zielplattformen, die aus deklarativen Beschreibungen zur Laufzeit konkrete Benutzungsschnittstellenelemente erzeugen können. Für Plattformen, deren Laufzeitumgebung keine Beschreibungssprache vorsieht, kann das Verfahren potentiell ebenfalls angewendet werden (z.B. zur Erzeugung vom konkretem Code oder durch Verschiebung der FUI-Transformation in die Laufzeitumgebung). Dies wurde jedoch im Rahmen der Arbeit nicht abschließend untersucht. Hier sind zukünftig weitere Untersuchungen erforderlich, welche die Anwendbarkeit das Verfahrens auch für diese Anwendungskategorie validieren.

9. *Shared UIs*: Wiederverwendung in Dienst-Ökosystemen

Dieses Kapitel beschreibt einen Ansatz zur Wiederverwendung von Benutzungsschnittstellen für Dienste in einem Dienst-Ökosystem. Der Ansatz ermöglicht die Komposition von Benutzungsschnittstellen, die von beliebigen Anbietern unterschiedlicher Domänen zur gemeinschaftlichen Nutzung bereitgestellt und zur Laufzeit an beliebige Dienst-Implentierungen eines Dienst-Ökosystems angebunden werden können.

Im Verlauf des Kapitels wird die Architektur einer Infrastruktur dargestellt, welche die dezentrale Bereitstellung von Benutzungsschnittstellen, deren Generierung und die Anbindung an Dienste eines Dienst-Ökosystems ermöglicht. Hierfür wird eine auf Ontologien basierende Repräsentation für datenzentrierte Anwendungsmodelle vorgestellt, die sich einerseits zur dezentralen Bereitstellung und Komposition eignet, andererseits die automatisierte Anbindung an Dienste zur Laufzeit gestattet.

Beiträge des Kapitels: Der Beitrag des Kapitels besteht in einem neuartigen Ansatz zur gemeinschaftlichen Nutzung und Wiederverwendung von Benutzungsschnittstellen für Dienste eines Dienst-Ökosystems. Die Beiträge im Einzelnen bestehen in

- einer universellen Repräsentation datenzentrierter Anwendungsmodelle
- einem Ansatz zur Anbindung an Dienste eines Dienst-Ökosystems
- einer Architektur / Infrastruktur zur dezentralen Wiederverwendung und Verarbeitung kombinierbarer Anwendungen.

9.1 Zielsetzung

In den Grundlagen in Abschnitt 2.5 wurde das Prinzip der *SharedUIs* als Mittel zur Wiederverwendung und gemeinschaftlichen Nutzung von Benutzungsschnittstellen dargestellt. Diese werden hierbei durch beliebige Anbieter über eine verteilte Infrastruktur zur Verfügung gestellt und können von Dritten in eigene Anwendungen eingebettet und im Sinne einer Komposition angepasst werden. Die Drittanwendung kann hierbei frei entscheiden, welche Benutzungsschnittstellen ausgewählt, welche konkrete technologische Variante generiert und welche konkreten Dienst-Implementierungen angesprochen werden. Die für eine Lösung relevanten Forschungsfragen wurden in Abschnitt 1.3 unter **(FF.5)** zusammengefasst. Sie beinhalten, wie Anwendungsmodelle zur gemeinschaftlichen Nutzung bereitgestellt werden und wie eine Anbindung an Dienste eines Ökosystems generisch erfolgen kann.

Abschnitt 2.5, Abbildung 2.10 zeigt das *Shared UI*-Prinzip an einem Beispiel (Reisebuchung mit Auswahl optionaler Komponenten). In Abbildung 9.1 ist das **Nutzungsszenario** verallgemeinert dargestellt: eine Dienst-Nutzer wählt aus einem Pool gemeinschaftlich nutzbarer Benutzungsschnittstellen (*Shared UIs*) für Ökosystem-Dienste aus ① und kombiniert diese in einer Oberfläche, die entsprechend dem technologischen Kontext der Anwendung generiert wird ②. Benutzer erfassen die Daten in den einzelnen Komponenten, die abschließend aufbereitet und an Dienste konkreter Anbieter im Dienst-Ökosystem zur Verarbeitung weitergereicht werden ③.

Folgende Aufgaben muss eine Drittanwendung zur Einbindung erfüllen: (1) die **Auswahl**, Beschaffung und Kombination von *Shared UIs* der Dienste, (2) die **Generierung** und Darstellung finaler Benutzungsschnittstellen und (3) die **Anbindung**, Aufbereitung und Übermittlung erfasster Daten an die Dienste.

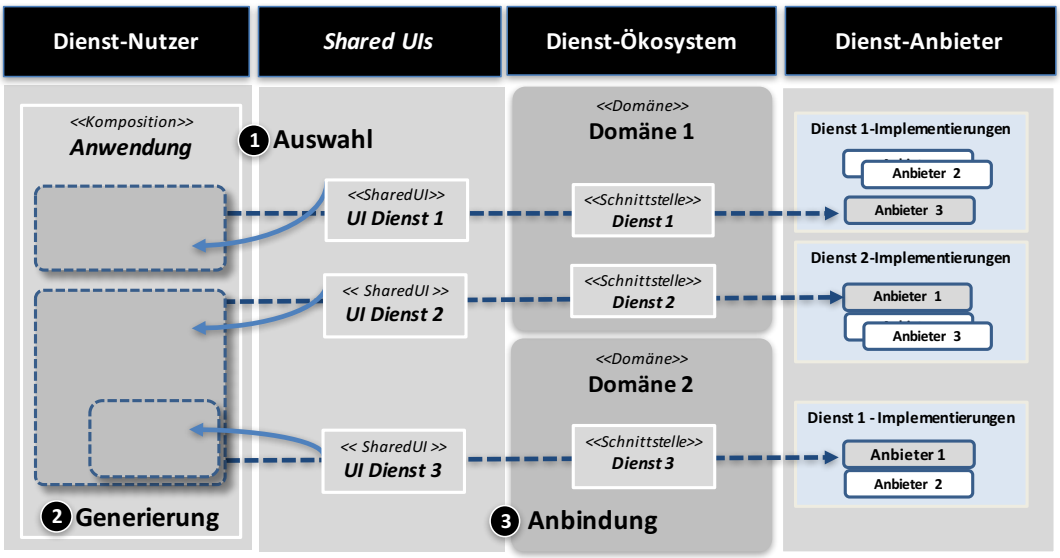


Abbildung 9.1. Integration von Diensten auf Benutzungsschnittstellenebene

Die Anforderungen an die Modellierung wurden in Abschnitt 3.1, Tabelle 3.1 unter **(A.4)** zusammengefasst. Sie bestehen darin, dass Anwendungsmodelle in einer **universellen Repräsentation** vorliegen, die von allen Teilnehmern im Ökosystem interpretiert werden kann **(A4.1)**. Darüber hinaus muss sie die **generische Anbindung an Dienste** im Dienst-Ökosystem ermöglichen **(A4.2)** und in einer **dezentralen Infrastruktur (A4.3)** genutzt werden können.

Das Ziel des Kapitels ist die Erweiterung des in den vorangegangenen Kapiteln entwickelten Ansatzes um das *Shared UIs*-Konzept und dessen Anwendung auf ein Dienst-Ökosystem. Im weiteren Verlauf werden folgende Erweiterungen vorgestellt:

- ein Architekturmuster für eine **dezentrale Infrastruktur** zur gemeinschaftlichen Nutzung von Anwendungsmodellen für Dienste (Abschnitt 9.3)
- eine **universellen Repräsentation von Anwendungsmodellen** (Abschnitt 9.4)
- ein Ansatz zur Modellierung zur **generischen Dienst-Anbindung** erforderlicher Informationen (Abschnitt 9.5)

Abschnitt 9.2 stellt zunächst den Lösungsansatz im Überblick vor.

9.2 Lösungsansatz

Das in Abschnitt 3.3 dargestellte Realisierungskonzept schlägt ein **Benutzungsschnittstellen-Ökosystem** vor, dessen Kern datenzentrierte Anwendungsmodelle bilden. Diese werden in einer universell interpretierbaren Repräsentation beschrieben. Sie beinhalten alle Informationen zur Herleitung der Benutzungsschnittstellen sowie zur Abbildung erfasster Daten auf Dienst-Schnittstellen.

Universelle Repräsentation des Anwendungsmodells. Die Voraussetzung, dass Artefakte wiederverwendet und von Drittanwendungen verarbeitet werden können, ist deren nicht-proprietäre Beschreibung (A4.1). Der vorgeschlagene Lösungsansatz verwendet semantische Technologien zur Beschreibung der Artefakte und schafft damit eine einheitliche *lingua franca* für das Benutzungsschnittstellen-Ökosystem. Das Realisierungskonzept (vgl.Abschnitt 3.3.3) besteht darin, die Benutzungsschnittstelle in Form einer Ontologie zu beschreiben. Hierfür wird das Anwendungsmodell in eine **Applikations-Ontologie (AO)** überführt, die mit standardisierten Mitteln (RDF/OWL) beschrieben wird.

Generische Dienst-Anbindung. Anwendungsmodelle werden zur Erfassung der Daten erstellt, die ein Dienst zur Durchführung eines Geschäftsprozesses benötigt. Um die in der Benutzungsschnittstelle erfassten Daten einem Dienst übergeben zu können, müssen sie in einer von der Dienst-Schnittstelle erwarteten Form vorliegen (z.B. Inhalte, Struktur, Format). Zur Beschreibung der Dienst-Schnittstelle werden im vorgeschlagenen Ansatz (vgl. Abschnitt 3.3.3) ebenfalls Ontologien verwendet. Eine **Dienst-Ontologie (DO)** beschreibt die Eingabedaten hinsichtlich ihrer Semantik und Struktur. Die Applikations-Ontologie für einen Dienst besitzt einen engen Bezug zu dieser Dienst-Ontologie, da sie basierend auf den darin beschriebenen Datenelementen aufgebaut wird. Die in der Applikations-Ontologie enthaltenen Daten korrelieren mit den Daten-Elementen der Dienst-Ontologie. Der Lösungsansatz besteht darin, die **Korrelation zwischen Elementen der Dienst-Ontologie und der Applikations-Ontologie** zu beschreiben und damit eine generische Abbildung der erfassten Daten auf die Dienst-Schnittstelle zu ermöglichen. Die Eingabedaten für die Dienst-Schnittstelle sind automatisch erzeugbar und können an eine konkrete Dienst-Implementierung übergeben werden (A4.2).

Aufbau einer dezentralen Infrastruktur. Das Realisierungskonzept zur gemeinschaftlichen Nutzung dieser Artefakte liegt in der Schaffung eines dezentralen Benutzungsschnittstellen-Ökosystems. Aufbauend auf dem Anwendungsmodell können generalisierte Komponenten erstellt werden, welche Drittanwendungen (1) die Auswahl von Anwendungsmodellen, (2) die Generierung einer Benutzungsschnittstelle und (3) die Aufbereitung der Eingaben zur generischen Dienst-Anbindung im Dienst-Ökosystem ermöglichen (vgl. Abschnitt 3.3.2).

9.3 Architektur eines dezentralen Benutzungsschnittstellen-Ökosystems

Abbildung 9.2 zeigt die Architektur eines solchen Benutzungsschnittstellen-Ökosystems, welches aus den Komponenten der vorangegangenen Kapitel aufgebaut ist.

Das zentrale Element ist das Anwendungsmodell, welches über eine **Applikations-Ontologie (AO)** repräsentiert wird und mit **Dienst-Ontologien (DO)** korreliert. Das Benutzungsschnittstellen-Ökosystem sieht folgende Komponenten vor, die für die Aufgabenstellungen der Auswahl, Generierung und Aufbereitung erfasster Daten zur Weiterleitung benötigt und als Dienste (z.B. Web-Services) bereitgestellt werden:

- **Anwendungsmodell-Repositories** Ⓐ ermöglichen die Auswahl von Anwendungsmodellen und Variantenbeschreibungen für Dienste. Die Modelle werden als Applikations-Ontologien zur Verfügung gestellt, die zusätzlich die Korrelationsinformationen zur Dienst-Ontologie enthalten.

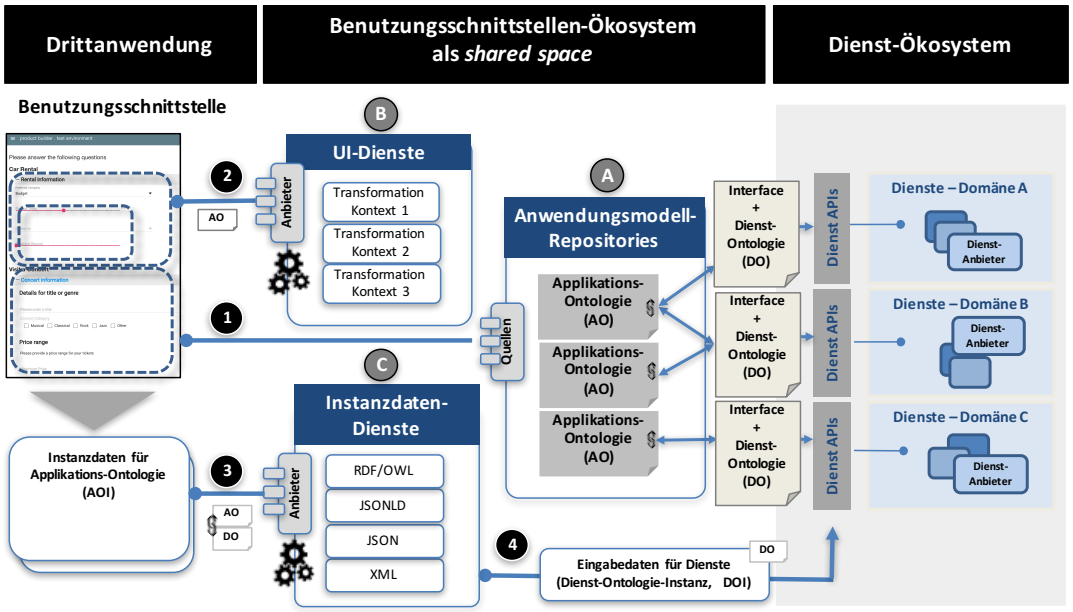


Abbildung 9.2. Verteiltes Benutzungsschnittstellen-Ökosystem

- **UI-Dienste** Ⓑ stellen Transformatoren zur Generierung von Benutzungsschnittstellen aus Anwendungsmodellen für unterschiedliche Interaktions- und Technologiekontexte bereit. Sie erhalten als Eingabe eine Applikations-Ontologie und erzeugen daraus die finale Benutzungsschnittstelle.

- **Instanzdaten-Dienste** Ⓒ stellen Transformatoren zur generischen Aufbereitung erfasster Daten (Instanzdaten) in Eingabedaten für einen konkreten Dienst bereit. Sie verwenden die Applikations-Ontologie und die Instanzdaten, um daraus die von der Dienst-Schnittstelle geforderten Eingabedaten zu erzeugen.

Die Applikations-Ontologien können dabei von beliebigen Anbietern zur Verfügung gestellt und damit gemeinschaftlich genutzt werden. Die funktionalen Komponenten ⒶⒷⒸ hängen lediglich von den Inhalten des Anwendungsmodells ab, welches alle Informationen zu den Daten und zur Generierung enthält. Da dieses in einer universell interpretierbaren Form beschrieben ist, können die Komponenten ebenfalls von verschiedenen Anbietern als Beitrag zum Ökosystem zur Verfügung gestellt werden.

Nutzungsszenario zur gemeinschaftlichen Nutzung

Das o.g. Nutzungsszenario lässt sich mit dieser Lösung umsetzen: eine Drittanwendung (vgl. Abbildung 9.2, links) kann in diesem Ökosystem aus beliebigen Quellen (**Anwendungsmodell-Repositories**) Anwendungsmodelle für Dienste beziehen und diese kombinieren ①. Zur Generierung der Benutzungsschnittstelle kann ein **UI-Dienst** ② bestimmt werden, der aus den Anwendungsmodellen eine einheitliche, auf den Kontext der Drittanwendung zugeschnittene Benutzungsschnittstelle generiert. Hierfür kann ein beliebiger *Anbieter* gewählt werden, der eine für den geltenden Interaktions- und Technologiekontext passende technische Variante erzeugen kann.

Nach erfolgter Eingabe kann ein **Instanzdaten-Dienst** gewählt werden, der aus den eingegebenen Daten einer Applikations-Ontologie eine Dienst-Schnittstellen-konforme Datenrepräsentation generiert ③. Das Ergebnis ist eine Menge von **Dienst-Ontologie-Instanzen (DOI)**, die von der Drittanwendung verarbeitet und an eine konkrete Dienst-Implementierung im Dienst-Ökosystem zur Weiterverarbeitung übergeben werden kann ④

9.4 Beschreibung des Anwendungsmodells als Ontologie

In den Arbeiten von Khushraj et al. [115] und Gaulke et al. [76] werden annotierte Ontologien zur Beschreibung von grafischen Benutzungsschnittstellen verwendet und damit eine universelle Beschreibung für diese Anwendungskategorie geschaffen. Der hier verfolgte Ansatz lehnt sich an diese Arbeiten an. Er verwendet ebenfalls **annotierte OWL-Ontologien**, unterscheidet sich jedoch im Inhalt der enthaltenen Informationen.

Während die genannten Ansätze auf die Erstellung grafischer Benutzungsschnittstellen fokussieren, werden hier die Inhalte des datenzentrierten Anwendungsmodells verwendet. Die Idee besteht darin, den im Anwendungsmodell beschriebenen **Strukturbaum als RDF-Graph** zu repräsentieren und dessen Knoten mit den zur Herleitung einer Benutzungsschnittstelle benötigten Informationen über Annotationen anzureichern. Hierzu werden die im Metamodell enthaltenen Informationen zu *Struktur/Aufbau*, *Typisierter Ein-/Ausgabe* und *Verhalten* in eine RDF/OWL-Repräsentation überführt und damit eine dem Anwendungsmodell äquivalente **Applikations-Ontologie** abgeleitet. Abbildung 9.3 zeigt dies am Beispiel der Adressdaten einer Antragstrecke. Das Ziel ist es, den Strukturbaum eines Anwendungsmodells (links) vollständig auf einen RDF-Graphen (rechts) abzubilden.

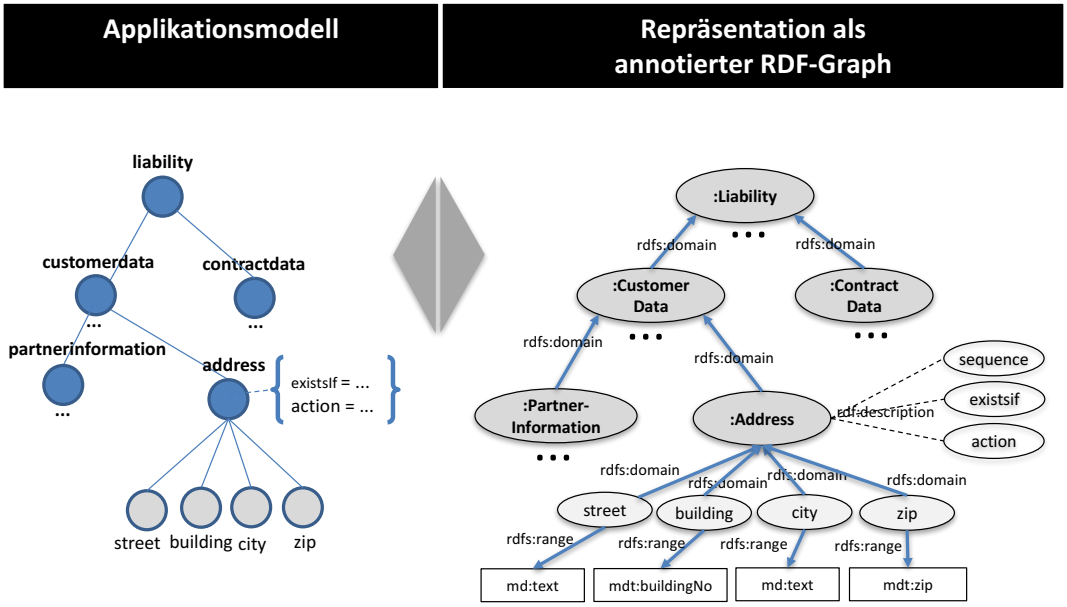


Abbildung 9.3. Repräsentation des Strukturbaums als RDF-Graph

Im Folgenden wird die Abbildung des Anwendungsmodells auf eine OWL-Ontologie zusammengefasst beschrieben. In Anhang A.5.1 wird dies vertieft und dargestellt, welche OWL-Standardkonstrukte verwendet wurden sowie das im Rahmen der Arbeit verwendete Annotations-Profil dargestellt.

9.4.1 Repräsentation des Strukturbaums als annotierter RDF-Graph

In der Informatik werden Ontologien dazu verwendet, Entitäten einer Domäne, deren Eigenschaften und Beziehungen semantisch zu beschreiben [95]. OWL als formale Beschreibungssprache für Ontologien bietet daher entsprechende Konstrukte zur Beschreibung von Entitäten (*OWLClasses*), Daten (*OWLDatatypeProperties*), deren Beziehungen und Eigenschaften (*OWLObjectProperties*). Das datenzentrierte Anwendungsmodell beschreibt analog Entitäten als Gruppen von Datenelementen, die in einer Beziehung zueinander stehen und bestimmte semantische Eigenschaften aufweisen. Die Abbildung des Anwendungsmodells auf eine OWL-Ontologie ist daher einfach möglich. Insbesondere die strukturellen Eigenschaften des Anwendungsmodells (Tabelle 5.1, (IR.1)) können direkt in eine OWL-Repräsentation überführt werden.

Abbildung 9.4 zeigt exemplarisch die Abbildung der im Anwendungsmodell enthaltenen Informationen auf Basiselemente einer OWL-Ontologie. Sie zeigt den Graphen aus Abbildung 9.3 in Turtle-Notation. Die Gruppen des Anwendungsmodells werden auf *owl:Class*-Elemente (Abbildung 9.4, ①) und Datenelemente auf *owl:DatatypeProperties* abgebildet ②. Die hierarchischen Beziehungen zwischen Gruppen werden über *owl:ObjectProperties* ③ beschrieben, welche Entitäten des Modells mit anderen Entitäten assoziieren. Die Struktur des Anwendungsmodells wird damit direkt auf einen RDF-Graphen abgebildet.

Jedoch können nicht sämtliche im Anwendungsmodell enthaltenen Informationen mit diesen RDF/OWL-Standardmitteln beschriebenen werden. So enthalten OWL-Ontologien üblicherweise keine Informationen zur *Sequenz* von Daten oder zur Formulierung von *Existenzbedingungen*, die auf Instanzdaten spezifiziert werden können. Zudem existieren nach meinem besten Wissen keine OWL-Konstrukte zur deklarativen Modellierung von *Verhalten*, welches ebenfalls auf Instanzdaten zur Laufzeit beschrieben werden muss. Um diese Informationen in der OWL-Ontologie abzubilden, wird das in OWL enthaltene Konzept der **Annotationen** verwendet. Khushraj et al. [115] und Gaulke et al. [76] verwenden Annotationen, um eine bestehende Ontologie für ein Daten- bzw. Taskmodell um Informationen für die Erstellung grafischer Benutzungsschnittstellen zu erweitern.

Analog der Beschreibung des Anwendungsmodells über eine DSL (Anhang A.3.1), werden die noch fehlenden Informationen als Zusatzinformationen den Gruppen und Datenelementen hinzugefügt. Dies erfolgt über Annotationen, die in einem **Annotationsprofil** [76] zusammengefasst und spezifiziert werden.

Abbildung 9.4 ④ zeigt dies exemplarisch anhand der Annotationen zur Beschreibung der Sequenz von Elementen (*ma:sequence*), der Kohäsion (*ma:cohesion*), typrelevanter Informationen (*ma:type*, *ma:restrictedTo, ma:initialValue, ma:semanticTags*) sowie zur Angabe des Verhaltens (*ma:existsIf, ma:validations, ma:reactions*).

Die verwendeten Annotationen und deren Nutzung entsprechen den Attributen, die bereits in der DSL zur Beschreibung von Gruppen und Datenelementen verwendet wurden. In Anhang A.3.1, Tabelle A.13 ist das verwendete Annotationsprofil dargestellt.

9.4.2 Bewertung der Lösung

Die Verwendung von **annotierten OWL-Ontologien** führt zu einer universellen Beschreibung des Anwendungsmodells, wie es in **(A4.1)** gefordert wird. Das Modell ist dadurch in

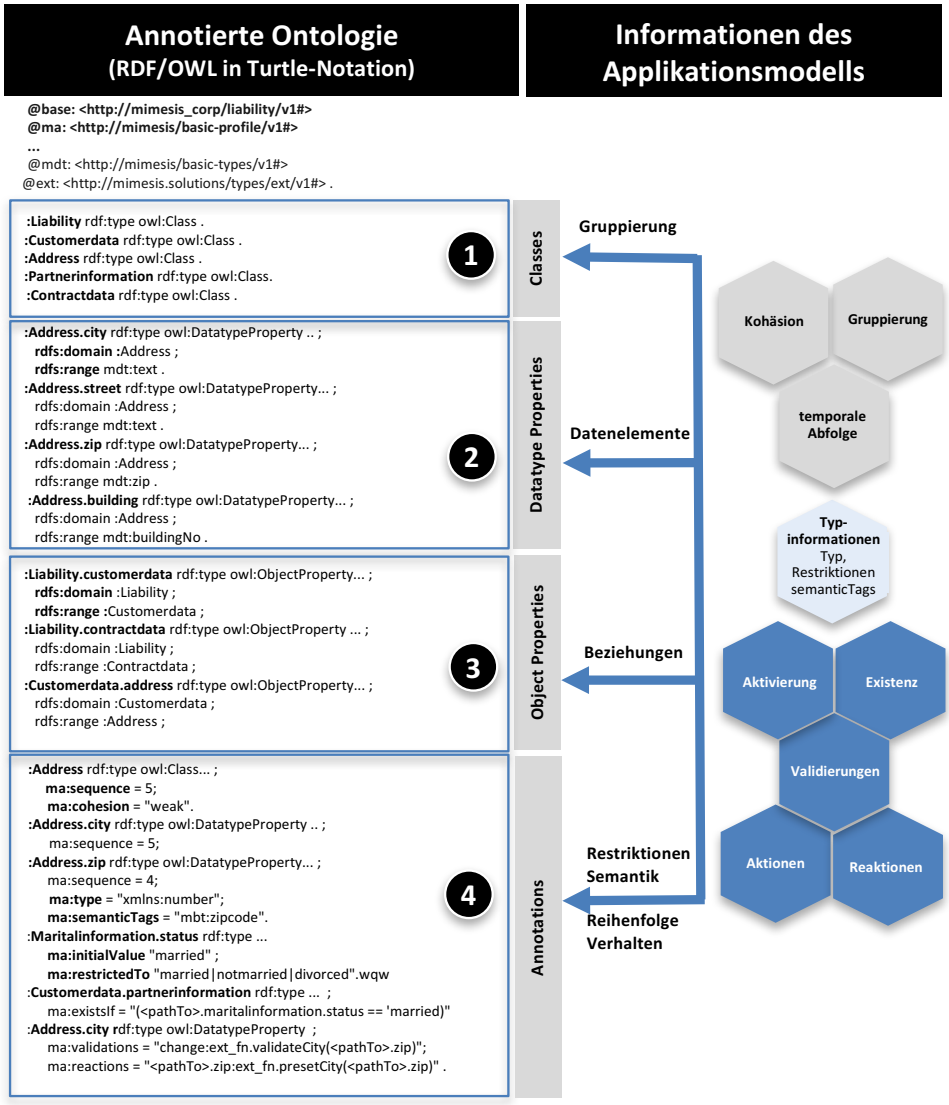


Abbildung 9.4. Abbildung der Metamodellinhalte auf OWL-Konstrukte

einer nicht-proprietären Form beschrieben und kann gemeinschaftlich genutzt werden. Zur Beschreibung der Inhalte des Anwendungsmodells werden lediglich Standardkonstrukte von RDF/OWL-Ontologien verwendet, wodurch Basisinformationen des Modells universell zugänglich sind. Durch die Verwendung von **Annotationen**, können darüber hinausgehende Informationen des Anwendungsmodells der Ontologie hinzugefügt werden. Das Resultat ist danach im Sinne der Generierung **vollständig spezifiziert**. Dritte, die das Profil kennen, sind in der Lage, diese Informationen z.B. zur Generierung einer Benutzungsschnittstelle auszuwerten. Nutzer ohne Kenntnis des Annotationsprofils können dennoch die Basisinformationen aus dem Modell ablesen und verarbeiten.

Die vorgeschlagene Ontologie stellt eine direkte Umsetzung des Metamodells (Kapitel 6) dar und beinhaltet alle dort enthaltenen Informationen zu *Struktur/Aufbau* (IR.1, IR.2), *Typisierten Ein-/Ausgabe* (IR.5-IR.7) und des *Verhaltens* (IR.10-IR.14). Es eignet es sich daher zur automatischen Generierung von Benutzungsschnittstellen unter Verwendung des in Kapitel 8 beschriebenen Verfahrens. Es erfüllt damit alle in (A4.1) gestellten Anforderungen.

Einschränkungen und zukünftige Arbeiten. Obwohl mit der dargestellten Umsetzung des Metamodells in eine Ontologie alle Eigenschaften in universeller Form beschrieben werden können, ist der Ansatz dennoch als erste Näherung zu werten. Die aktuelle Umsetzung weist noch Einschränkungen auf, die ggf. vermieden werden können.

Die dargestellte Modellierung orientiert sich stark an der klassischen Modellierung von Daten, wie sie beispielsweise in der objekt-orientierten Modellierung angewendet wird. Obwohl dies legitim ist, findet sich in der Literatur Kritik an dieser Vorgehensweise, die ggf. zukünftig eine optimierte Modellierung nahelegen[48]. Die dargestellte Umsetzung verwendet zudem überwiegend Annotationen zur Modellierung von Aspekten der Benutzungsschnittstelle. Dies schränkt die Universalität der Modellierung ein, da bestimmte Eigenschaften versteckt in Annotationen liegen, die ggf. mit Standardmitteln darstellbar sind. Hier ist in zukünftigen Arbeiten zu prüfen, ob die Verwendung weiterer OWL-Konstrukte den Ansatz allgemeiner gestalten können (z.B. Alternativen zur Modellierung von Existenzbedingungen).

Der vorgestellte Ansatz ist bewusst beschränkt auf hierarchische Strukturen zur Gruppierung von Informationen in den Ontologien. Aus Sicht dialogbasierter Anwendungen ist das keine Einschränkung, da diese *per se* auf hierarchischen Datenstrukturen arbeiten. Diese Charakteristik beschränkt jedoch die Übertragung des Ansatzes auf beliebige Ontologien, welche eine netzartige Struktur aufweisen können. Sahar et al. [197] untersuchen dieses Problem im Umfeld der UI-Generierung für beliebige Ontologien, beschränken sich jedoch auf die Erzeugung sehr einfach gearteter UIs. Die Erkenntnisse dieser Arbeit können in zukünftige Erweiterungen des datenzentrierten Ansatzes einfließen.

9.5 Anbindung der Benutzungsschnittstelle an Ökosystem-Dienste

Um eine Benutzungsschnittstelle an Dienste in einem Dienst-Ökosystem anbinden zu können, müssen die erfassten Daten in einer für den Dienst verständlichen Form aufbereitet werden. Diese sind in einer **Dienst-Ontologie (DO)** beschrieben, die Struktur und Semantik der Eingabedaten festlegt. Eine **Applikations-Ontologie (AO)** zur Beschreibung der Benutzungsschnittstelle basiert auf dieser Dienst-Ontologie und bereitet die Eingabedaten zur Präsentation auf, indem sie in eine für den Benutzer sinnvolle Dialogstruktur gebracht werden.

Abbildung 9.5 (rechts) stellt dies am Beispiel der Kundendaten einer Antragstrecke dar. Auf der rechten Seite ist die **Struktur einer Instanz der Dienst-Ontologie** abgebildet. Die zugrundegelegte, exemplarische Dienst-Ontologie (*lro:...liability/v1.0*) ist zur Illustration so konstruiert, dass die Daten in einer flachen Hierarchie beschrieben sind. Sie fasst die Antragsdaten in einer Entität zusammen (*:liabilityRequest*). Links ist die Struktur der **Instanzdaten des Anwendungsmodells** dargestellt. Sie unterscheidet sich von der Zielstruktur, obwohl es sich um dieselben Informationen handelt.

Wenn der Nutzer zur Laufzeit Daten erfasst, baut er damit **eine Instanz der Applikations-Ontologie (AOI)** auf. Da die Applikations-Ontologie für eine **spezifische Dienst-Ontologie** erstellt wurde, korrelieren die erfassten Informationen (AOI-Gruppen, AOI-Datenelemente) eindeutig mit Elementen der Dienst-Ontologie. Abbildung 9.5 stellt diese Beziehung dar. AOI-Gruppen korrelieren mit *Entitäten*, AOI-Datenelemente mit *DatatypeProperties* der

[48] z.B. Diskussion in RDF Primer 1.0, 2004. Abschnitt 5.3. https://www.w3.org/TR/rdf-primer/

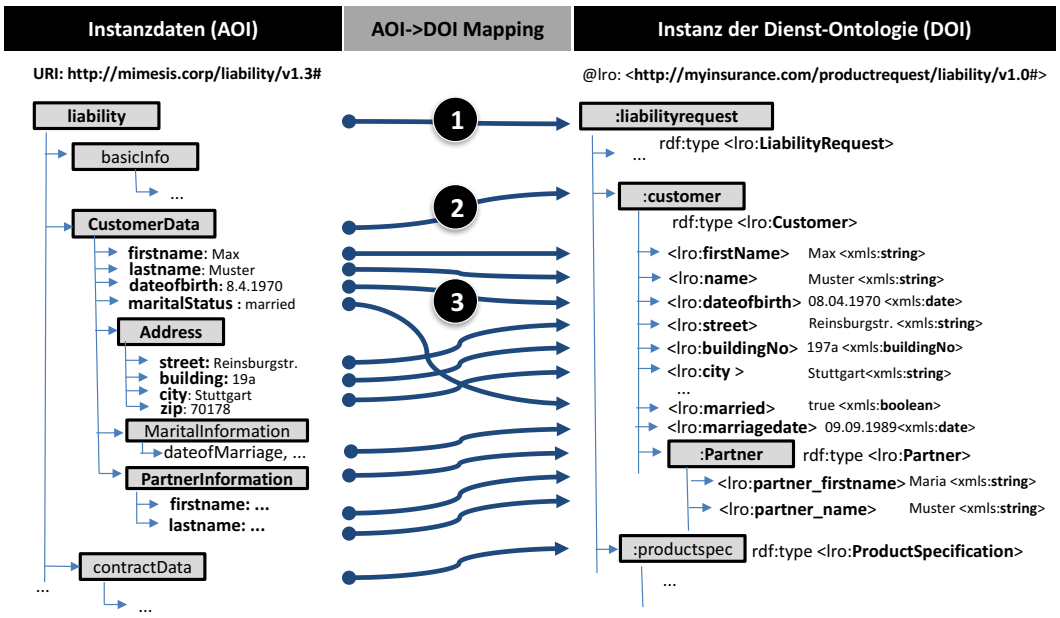


Abbildung 9.5. Abbildung Instanzdaten auf die Zielontologie

Dienst-Ontologie. Die Abbildung zeigt, dass sich die Struktur der Daten in beiden Sichten unterscheiden kann. So erscheinen z.B. die in den Instanzdaten in einer gesonderten Gruppe zusammengefassten *Adressdaten* in der Zielstruktur auf der selben Ebene mit den *Namensinformationen*.

Sollen die Instanzdaten an einen Dienst zur Verarbeitung übergeben werden, müssen sie in eine **Instanz der Dienst-Ontologie (DOI)** überführt werden. Die Hauptaufgabe besteht darin, die Elemente der AOI auf Elemente der DOI abzubilden. Diese Transformation kann durch ein einfaches Strukturmapping erreicht werden. Hierzu müssen im Anwendungsmodell die Informationen zur Korrelation zum durch die Dienst-Ontologie spezifizierten Zielmodell enthalten sein.

Beschreibung der Korrelation zur Dienst-Ontologie. Die Beschreibung der Korrelation der Elemente umfasst zwei Aspekte:

- **semantische Korrelation:** Welche Semantik besitzt eine Gruppe / ein Datenelement in der Dienst-Ontologie?
- **strukturelle Korrelation:** Wo befindet sich das Element in der Zielstruktur?

Durch die **semantische Korrelation** wird für Gruppen bzw. Datenelemente der Instanzdaten beschrieben, welche Semantik sie in der DOI besitzen. Hierzu muss für eine Gruppe die korrelierende Entität und dessen Klasse und für ein Datenelement das korrelierende Daten-Property und dessen Typ angegeben werden. In Abbildung 9.5 ① erfolgt z.B. die Abbildung der AOI-Gruppe *Liability* auf eine Instanz der Klasse *lro:LiabilityRequest* und die Kundendaten (*CustomerData*) auf eine Instanz von *lro:Customer* ②. Das Datenelement *dateOfBirth* wird auf ein Element *:dateofbirth* mit dem Typ *xml:date* abgebildet ③.

Die **strukturelle Korrelation** beschreibt, wo sich eine Gruppe bzw. ein Datenelement in der DOI wiederfindet. Sowohl für Gruppen- als auch Datenelemente erfolgt dies durch Zuordnung einer Gruppe / eines Datenelements zu einer Entitäts-Instanz in der Zielstruktur der

DOI. In Abbildung 9.5 ist dies über die hierarchische Struktur dargestellt. Die Instanz *:customer* ist Teil des Anfrageobjekts und erfordert daher eine Zuordnung zu *:liabilityrequest*. Die *:partner*-Instanz wiederum ist Teil der *:customer*-Instanz. Analog sind die Datenelemente Entitäts-Instanzen zugeordnet. Z.B. ist das Datenelement *:dateofbirth* Teil der *:customer*-Instanz.

Liegen die Korrelationsinformationen zu den Elementen der Instanzdaten vor, kann daraus eine Instanz der Ziel-Ontologie-Instanz erstellt werden. Diese ist aus Elementen der Dienst-Ontologie aufgebaut und repräsentiert die Instanzdaten in der von der Dienst-Schnittstelle geforderten Struktur.

9.5.1 Darstellung der Korrelation in der Applikations-Ontologie

Zur Beschreibung der Korrelation wird wiederum das Konzept der **annotierten Ontologien** angewendet. Hierzu werden die *owl:Class*, *owl:ObjectProperties* und *owl:DatatypeProperties* der Applikations-Ontologie mit Annotationen zur Korrelation angereichert. Die Annotationen beschreiben für die vorhandenen Gruppen und Datenelemente, welchen Elementen sie in der Dienst-Ontologie entsprechen (semantische Korrelation) und in welche Struktur die erfassten Instanzdaten überführt werden müssen (strukturelle Korrelation). Das hierfür verwendete Annotations-Profil ist in Anhang A.5.2, Tabelle A.14 dargestellt. Die Tabelle zeigt die Annotationen zur Beschreibung der semantischen und strukturellen Korrelation und gibt an, auf welche Elemente der Applikations-Ontologie sie anwendbar sind.

Abbildung 9.6 zeigt exemplarisch die Verwendung der Annotationen für ausgewählte Elemente der Applikations-Ontologie aus Abschnitt 9.4. Die enthaltenen Elemente werden zur Beschreibung der Korrelation mit Annotationen des Korrelations-Profils ergänzt[49].

Die Abbildung des **Wurzelelements der Applikations-Ontologie** (*Liability*) erfolgt auf eine Instanz (*:liabilityrequest*) der Klasse *lro:LiabilityRequest* ①. Die Instanz wird dabei über die *sa:swIndividual*-Annotation, die Klasse über *sa:swClass* angegeben. Analog werden die **Gruppen** *:Liability.customerdata* und *:Customerdata.partnerinformation* auf Klassen der Dienst-Ontologie abgebildet ②. Zusätzlich wird hier eine strukturelle Korrelation beschrieben (*sa:swForIndividual*, *sa:swProperty*), die das *:customer*-Objekt der *:liabilityrequest*-Instanz und das *:partner*-Objekt der *:customer*-Instanz zuordnet. Das Mapping für **Datenelemente** erfolgt über die Angabe des Typs (*sa:swType*) und der struturellen Korrelation (*sa:swForIndividual*, *sa:swProperty*) zur Dienst-Ontologie ③. Die Datenelemente werden darüber mit einer Entitäts-Instanz verbunden.

9.5.2 Erzeugung einer Dienst-Ontologie-Instanz

Zur Laufzeit kann aus den Korrelationsinformationen eine Dienst-Ontologie-Instanz erzeugt werden. Abbildung 9.6 (rechts) zeigt die Dienst-Ontologie-Instanz in Form eines Graphen, der sich aus den Korrelationsinformationen ableitet. Zur Erstellung des Graphen zur Laufzeit müssen lediglich für alle erfassten Gruppen und Datenelemente der Applikations-Ontologie

[49] Bereits vorhandene Annotationen des Basis-Profils sind aus Gründen der Übersichtlichkeit in Abbildung 9.6 nicht dargestellt.

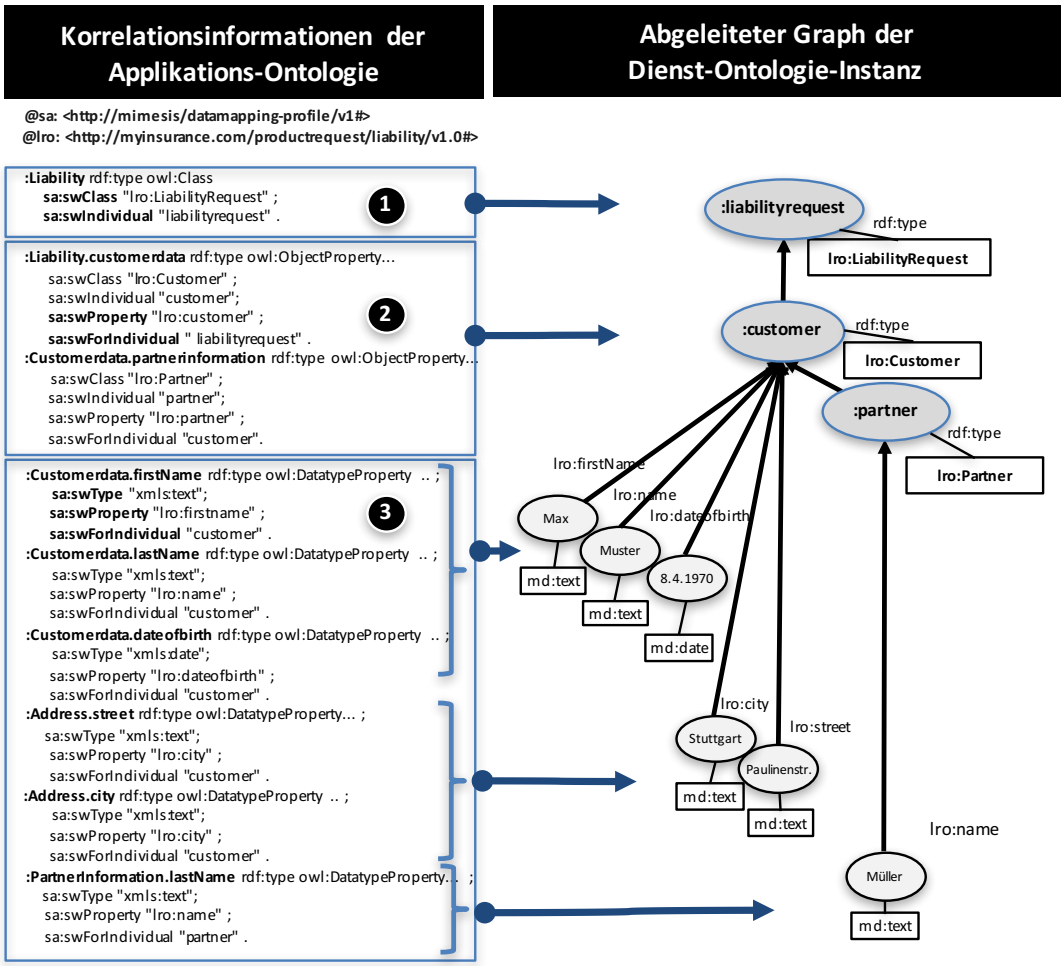


Abbildung 9.6. Anwendung der Annotationen des Korrelations-Profils in der Applikations-Ontologie

entsprechende Knoten der Dienst-Ontologie-Instanz erzeugt und mit den Instanzdaten belegt werden.

In Anhang A.5.3 wird ein einfaches Verfahren beschrieben, welches diese Schritte durchführt. Das dort dargestellte Verfahren benötigt lediglich das Anwendungsmodell mit den angefügten Korrelationsinformationen, um für gegebene Instanzdaten die Dienst-Ontologie-Instanz zu erzeugen. Damit kann nach der Dateneingabe durch den Nutzer **zur Laufzeit eine Dienst-Ontologie-Instanz abgeleitet werden**, die als Eingabe für einen Dienst im Dienst-Ökosystem benötigt wird.

9.5.3 Bewertung der Lösung

Die dargestellte Beschreibung der Korrelationen zwischen Elementen der Applikations- und der Dienst-Ontologie ermöglichen die automatische Herleitung von Dienst-Ontologie-Instanzen aus Instanzdaten. Der Modellierungsansatz ermöglicht die Erstellung generischer Transformatoren als Instanzdaten-Dienste in einer Architektur (Abbildung 9.2), die dezentral durch verschiedene Anbieter zur Verfügung gestellt werden können. Die generierten Dienst-Ontologie-Instanzen können in ein standardisiertes Format überführt und an eine beliebige

Dienst-Implementierung zur Verarbeitung übergeben werden. Das beschriebene Vorgehen erfüllt damit die Anforderung **(A4.2)**.

Einschränkungen und zukünftige Arbeiten. In der dargestellten Form verwendet der Ansatz eine stark vereinfachte Methode, die lediglich eine **unidirektionale Abbildung von Instanzdaten auf eine Dienst-Ontologie-Instanz** erlaubt. Die Korrelationsinformationen und dargestellten Verfahren beschränken sich bisher lediglich auf die AOI-zu-DOI-Transformation. Eine mögliche Rücktransformation von DOI auf AOI Daten, wie sie z.B. zur Vorbelegung von Daten einer Benutzungsschnittstelle verwendet werden kann, ist damit nur eingeschränkt möglich. Um dies in zukünftigen Erweiterungen des Ansatzes zu ermöglichen, kann jedoch auf bestehende Arbeiten im Bereich bidirektionaler Baumtransformationen (*bi-directional tree transformations*, z.B. Foster et al. [69]) zurückgegriffen werden.

Der vorgestellte Ansatz beschränkt sich bisher auf ontologische Repräsentationen. Er geht von der **Existenz einer ontologischen Beschreibung der Dienst-Schnittstellen** (DO) aus und betrachtet die konkrete Abbildung von Eingabedaten auf Dienst-Ontologie-Instanzen. Dies schränkt die Universalität des Verfahrens nur scheinbar ein, da die Abbildung prinzipiell auch für andere Technologien erfolgen kann (z.B. die Abbildung auf XML/WebService- oder JSON/REST-Schnittstellen). Dieser Aspekt wurde jedoch im Rahmen der Arbeit nicht umfassend untersucht. Da es sich bei der beschriebenen Transformation um eine einfache Struktur- und Typ-Transformation handelt, ist davon auszugehen, dass eine technologieneutrale Repräsentation der Korrelationsinformationen gefunden werden kann. In zukünftigen Arbeiten kann eine abstrahierte Beschreibung der Korrelationsinformationen in das Metamodell integriert werden.

9.6 Beispiel für *Shared UIs*

Im folgenden Beispiel wird anhand einer kombinierten Reisebuchungs-Anwendung (vgl. Abschnitt 2.5) die Nutzung von *Shared UIs* gezeigt. Die Reisebuchung besteht aus einer Rahmenanwendung zur Erfassung der allgemeinen Reise- und Kundendaten und bietet eine Reihe von Reise-Produkten zur Auswahl (z.B. Übernachtung, Flüge und Mietwagen), deren UIs als externe Komponenten eingebunden werden (s. Abbildung 9.7).

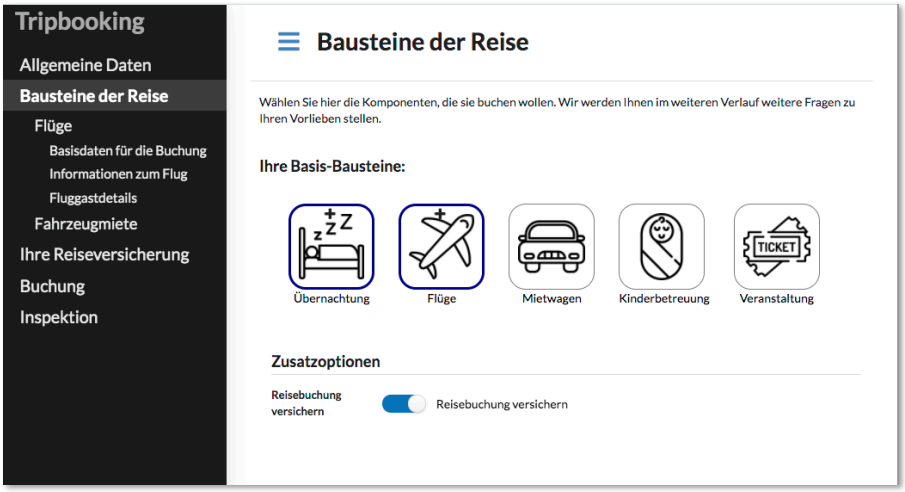


Abbildung 9.7. Auswahl von Reise-Produkten

Aus den erfassten Daten soll anschließend für jede der gewählten Komponenten eine Instanz der zugehörigen Dienst-Ontologie abgeleitet werden, die an konkrete Dienste zur Verarbeitung weitergeleitet werden kann.

In Anhang A.8.3 ist das Anwendungsmodell der Reisebuchung in *mimesis.DSL*-Notation angegeben. Es handelt sich um eine Komposition, welche die UIs für die einzelnen Reise-Produkte als Komponenten referenziert und anzeigt, sobald sie ausgewählt wurden. Die Referenzierung erfolgt über die Angabe der eindeutigen URI. Listing 9.1 zeigt die Referenzierung der Komponenten und die Existenzbedingungen zur Anzeige in der Rahmenanwendung.

```
...
component : "flightbooking"
{
    uri ="http://aviation.mimesis.solutions/products/flightbooking/mw2020"
    ...
    existsIf ="contains(tripbooking.options.components,'flight')"
}
component : "carrental"
{
    uri ="http://bookers.mimesis.solutions/products/carrental/mw2020"
    ...
    existsIf ="contains(tripbooking.options.components,'car')"
}
component : "overnightstay"
{
    uri ="http://bookers.mimesis.solutions/products/overnightstay/v1"
    ...
    existsIf="contains(tripbooking.options.components,'overnightstay')"
}
...
```

Listing 9.1: Shared UI-Referenzen im Anwendungsmodell

Im eingangs beschriebenen Nutzungsszenario (Abschnitt 9.3) fordert der Dienst-Nutzer von einem *Anwendungsmodell-Repository* das Modell für eine Reisebuchung an und übergibt es dem *UI-Dienst* zur Erzeugung einer finalen Benutzungsschnittstelle. Wie in der Darstellung des Generierungsverfahrens beschrieben, erkennt der UI-Dienst, dass es sich um eine Komposition handelt, löst die enthaltenen Komponenten-Referenzen auf und baut aus den gesammelten Modellen ein Variantenmodell, das zur Generierung der finalen UI genutzt und präsentiert wird. Die von der UI erfassten Daten müssen dann in Dienst-Ontologie-Instanzen der einzelnen Komponenten überführt werden.

Eine Benutzungsschnittstelle aus SharedUIs

Im dargestellten Benutzungsschnittstellen-Ökosystem können sowohl die Rahmenanwendung als auch die Anwendungsmodelle der Komponenten als Ontologien beschrieben werden. Da die ontologischen Darstellungen umfangreich sind, fokussieren sich die folgenden Darstellungen auf die Komponente zur Flugbuchung. Listing 9.2 zeigt fragmentarisch deren *Shared UI* als Applikations-Ontologie in Turtle-Notation. Eine detailliertere Version befindet sich in Anhang A.8.6.

Die in Listing 9.2 ① dargestellten Klassen in der Ontologie repräsentieren die Gruppen des Anwendungsmodells (z.B. *Flightbooking, Flightinfo, CustomerInfo*). Die Beziehungen zwischen den Gruppen werden durch *ObjectProperties* ② beschrieben. So ist z.B. *FlightInfo*

Teil von *Flightbooking* und nimmt dort die Rolle *Flightbooking.flightinfo* ein. In den Gruppen enthaltene Datenelemente werden über *DataProperties* ③ spezifiziert. Es wird angegeben, welchen Klassen sie zugeordnet sind (*rdfs:domain*) und welchen Basistyp sie besitzen (*rdfs:range*). Mit diesen RDF/OWL-Konstrukten wird die Struktur der Benutzungsschnittstelle spezifiziert.

```
@prefix : <http://aviation.mimesis.solutions/products/flightbooking/v1#> .
@prefix owl: ... rdf: ... xml:  ... xsd: ... rdfs: ...
@base <http://aviation.mimesis.solutions/products/flightbooking/v1> .
<http://aviation.mimesis.solutions/products/flightbooking/v1> rdf:type owl:Ontology .

❶ Classes
    :Flightbooking rdf:type owl:Class .
    :Basictraveldata rdf:type owl:Class .
    :Flightinfo rdf:type owl:Class .
    :Customerinfo rdf:type owl:Class .
    :Persons rdf:type owl:Class .
    :Flight rdf:type owl:Class .
    :Returnflight rdf:type owl:Class .
    :Openjawflightinfo rdf:type owl:Class .
    :Customer rdf:type owl:Class .
    :Address rdf:type owl:Class .
    ...
❷ Object Properties
    : Flightbooking.flightinfo  rdf:type owl:ObjectProperty ;
        rdfs:domain :Flightbooking ; rdfs:range :Flightinfo .
    :Flightinfo.flight rdf:type owl:ObjectProperty ;
        rdfs:range :Flight ; rdfs:domain :Flightinfo .
    :Flightbooking.basictraveldata rdf:type owl:ObjectProperty;
        rdfs:range :Basictraveldata ;  rdfs:domain :Flightbooking .
    :Flightbooking.customerinfo  rdf:type owl:ObjectProperty ;
        rdfs:range :Customerinfo ;  rdfs:domain :Flightbooking .
    :Basictraveldata.persons rdf:type owl:ObjectProperty ;
        rdfs:domain :Basictraveldata ;  rdfs:range :Persons .
    :Flightinfo.returnflight rdf:type owl:ObjectProperty ;
        rdfs:domain :Flightinfo ; rdfs:range :Returnflight .
    :Returnflight.openjawflightinfo rdf:type owl:ObjectProperty;
        rdfs:range :Openjawflightinfo ; rdfs:domain:Returnflight.
    ...
❸ Data Properties
    :Flight.fromdestination  rdf:type owl:DatatypeProperty ;
        rdfs:domain :Flight ; rdfs:range xsd:string .
    :Flight.todestination rdf:type owl:DatatypeProperty;
        rdfs:domain :Flight ; rdfs:range xsd:string .
    :Flight.startdate rdf:type owl:DatatypeProperty ;
        rdfs:domain :Flight ; rdfs:range xsd:date .
    :Flight.returnflight rdf:type owl: DatatypeProperty;
        rdfs:domain :Flight ; rdfs:range xsd:boolean .
    :Returnflight.returndate rdf:type owl:DatatypeProperty;
        rdfs:domain :Returnflight ; rdfs:range xsd:date .
    ...
```

Listing 9.2: Elemente der Flugbuchungs-Onotlogie

Da nicht alle Informationen des Anwendungsmodells mit Standard-Konstrukten von RDF bzw. OWL beschrieben werden können, werden darüber hinausgehende Eigenschaften als Annotationen beschrieben. Listing 9.3 zeigt Beispiele für Annotationen, die Informationen z.B. für Typ, Restriktionen, Sequenz oder Existenzbedingungen einzelnen Elementen der Ontologie hinzufügen.

```
:Flightinfo.flight
    ma:sequence "1" ;
:Flight.startdate
    ma:sequence "1" ;
    ma:type "date" ;
:Flightinfo.returnflight
    ma:existsIf "(returnflight == true)" ;
    ma:sequence "2" .
:Flight.returnflight
    ma:sequence "4" ;
    ma:type "boolean" ;
    ma:initialValue "true" .
:Returnflight.returndate
    ma:sequence "1" ;
    ma:type "date" ;
:Flight.fromdestination
    ma:sequence "2" ;
    ma:restrictedTo "flightbooking.getDepartureAirports()" ;
    ma:type "text" ;
:Flight.todestination
    ma:restrictedTo " " ;
    ma:sequence "3" ;
    ma:type "text" ;
    ma:activeIf "fromdestination.length0" ;
    ma:reactions "fromdestination:flightbooking
        .changeDestinations( fromdestination,$todestination)" .
:Returnflight.openjawflightinfo
    ma:existsIf "(openjawflight == true)" ;
    ma:sequence "3" .
...
```

Listing 9.3: Annotationen in der Flugbuchungs-Onotlogie

Das Resultat ist eine ontologische Beschreibung, die inhaltlich mit dem erarbeiteten Metamodell übereinstimmt. Der UI-Dienst kann daraus ein Modell der Anwendung erzeugen, die Komposition und Variantentransformation damit ausführen und mit dem vorgestellten Generierungsverfahren eine finale UI generieren.

Erzeugen von Dienst-Ontologie-Instanzen der Komponenten

Nach erfolgter Eingabe stehen die Instanzdaten der Reisebuchung zur Verfügung und müssen in Dienst-Ontologie-Instanzen für die integrierten Komponenten überführt werden. Listing 9.4 zeigt exemplarisch Instanzdaten in JSON-Notation, die im Verlauf der Nutzung für die Flugbuchungskomponente gesammelt wurden.

Zur Überführung der Eingaben in eine Dienst-Ontologie-Instanz sind in den Anwendungs-Ontologien Korrelations-Informationen hinterlegt. Diese sind wiederum als Annotationen für die Modellelemente angegeben. Abbildung 9.8 illustriert grafisch einen Teil des Mappings, welches für die Flugbuchungs-Komponente erreicht werden soll. Listing 9.5 zeigt den entsprechenden Auszug der Korrelations-Informationen, wie sie in der Anwendungs-Ontologie angegeben werden. Aus den Instanzdaten und den Korrelationsinformationen kann für die Flugbuchung dann eine Dienst-Ontologie-Instanz hergeleitet werden. Listing 9.6 zeigt einen Ausschnitt der Generierten Dienst-Ontologie-Instanz für die Flugbuchung.

Analog dem dargestellten Beispiel für die Flugbuchung wird für jede Komponente verfahren. Daraus entsteht eine Sammlung an Dienst-Ontologie-Instanzen, welche die erfassten Daten vollständig und zur Weiterverarbeitung geeignet beinhaltet.

```
"tripbooking": {
   ...
   "options": {
    "flightbooking": {
    "flightinfo": {
      "flight": {
        "fromdestination": "HAM",
        "todestination": "STR",
        "returnflight": true,
        "startdate": "2020-06-13T22:00:00.000Z"},
      "returnflight": {
        "openjawflight": true,
        "returndate": "2020-06-26T22:00:00.000Z",
        "openjawflightinfo": {
          "departuredate": "2020-06-22T22:00:00.000Z",
          "openyawfromdestination": "MUC",
          "openyawtodestination": "HAM"}}},
    "passengerinfo": {
      "youngpassengers": {
        "child1": {
          "age": "10"}},
      "adultpassengers": {
        "adult1": {
          "firstname": "Max",
          "lastname": "Mustermann"}}},
    "basictraveldata": {
      "preferences": {
        "placement": "window|isle",
        "category": "economy",
        "food": "vegetarian"},
      "persons": {
        "children": "1",
        "adults": "2"}},
    "customerinfo": {
      "fullname": {
        "firstname": "Max",
        "lastname": "Mustermann" ... }} ...
```

Listing 9.4: Instanzdaten der Reisebuchung (Flugbuchungs-Komponente)

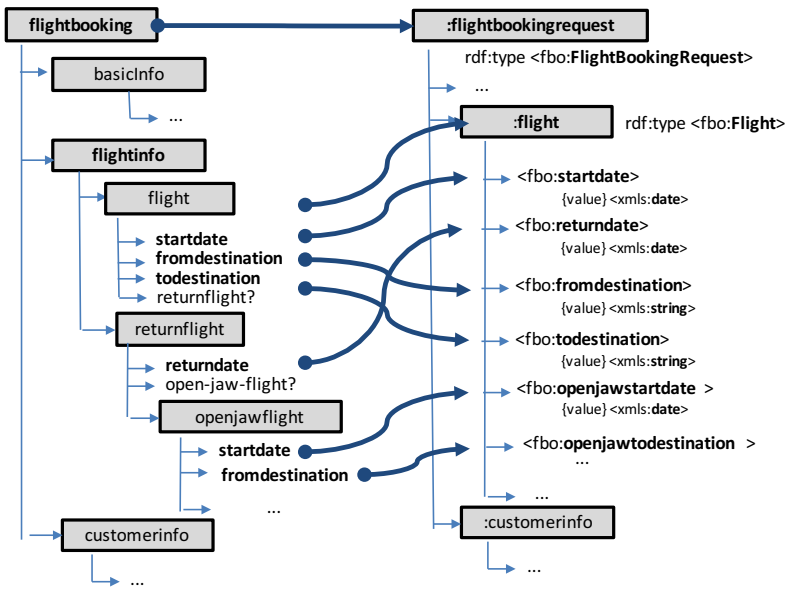


Abbildung 9.8. Mapping von Instanzdaten zur Dienst-Ontologie-Instanz

```
:Flightbooking.flightinfo
    sa:swClass "fbo:FlightBookingRequest" ;
    sa:swIndividual "flightbookingrequest" .
:Flightinfo.flight
    sa:swClass "fbo:Flight" ;
    sa:swProperty "fbo:flight" ;
    sa:swIndividual "flight" ;
    sa:swForIndividual "flightbookingrequest" .
 :Flight.startdate
    sa:swProperty "fbo:startdate" ;
    sa:swForIndividual "flight" ;
    sa:swType "xmls:date" .
 :Flight.fromdestination
    sa:swProperty "fbo:fromdestination" ;
    sa:swForIndividual "flight" ;
    sa:swType "xmls:string" .
... .
 :Returnflight.returndate
    sa:swProperty "fbo:returndate" ;
    sa:swForIndividual "flight" ;
    sa:swType "xmls:date" .
:Openjawflightinfo.openjawfromdestination
    sa:swProperty "fbo:openjawfromdestination" ;
    sa:swForIndividual "flight" ;
    sa:swType "xmls:string" .
:Openjawflightinfo.departuredate
    sa:swProperty "fbo:openjawstartdate" ;
    sa:swForIndividual "flight" ;
    sa:swType "xmls:date" .
```

Listing 9.5: Korrelations-Informationen der Flugbuchungs-Komponente

```
@prefix owl: ... ,rdf: ..., xml:..., xsd, ..., rdfs:...
@prefix fbo: <http://mimesis.solutions/bookers/flightbooking/v1#> .
@prefix foaf: <http://xmlns.com/foaf/0.1#> .
@base <http://aviation.mimesis.solutions/products/flightbooking/v1/individuals> .
ă<http://aviation.mimesis.solutions/products/flightbooking/v1/individuals>
     rdf:type owl:Ontology .

:flightbookingrequest_i1474371413428
    rdf:type <fbo:FlightBookingRequest> ,  owl:NamedIndividual ;
            <fbo:childtickets> "1"^^<xmls:number> ;
            <fbo:adulttickets> "2"^^<xmls:number> ;
            <fbo:customerinfo> :customerinfo_i1474371413428 ;
            <fbo:flight> :flight_i1474371413428 .
:flight_i1474371413428 rdf:type <Flight> ,owl:NamedIndividual ;
            <fbo:returndate> "2020-06-26T22:00:00.000Z"^^<xmls:date> ;
            <fbo:startdate> "2020-06-13T22:00:00.000Z"^^<xmls:date> ;
            <fbo:fromdestination> "HAM"^^<xmls:string> ;
            <fbo:todestination> "STR"^^<xmls:string>;
            <fbo:openyawstartdate> "2020-06-22T22:00:00.000Z"^^<xmls:date> ;
            <fbo:openyawtodestination> "HAM"^^<xmls:string> ;
            <fbo:openyawfromdestination> "MUC"^^<xmls:string> .
:customerinfo_i1474371413428 rdf:type <Customerinfo> , ... ;
            <foaf:givenName> "Max"^^<xmls:string> ;
            <foaf:familyName> "Mustermann"^^<xmls:string> ;
            <foaf:gender> "male"^^<xmls:string> ;
            <foaf:email> "max.mustermann@web.com"^^<xmls:string> ;
 ...
```

Listing 9.6: Generierte Dienst-Ontologie-Instanz der Flugbuchungs-Komponente (Auszug)

9.7 Diskussion der Ergebnisse

Das Ziel des Kapitels lag in der Darstellung des Ansatzes zur gemeinschaftlichen Nutzung und Wiederverwendung von Benutzungsschnittstellen für Dienste eines Dienst-Ökosystems (*Shared UIs*). Mit der Lösung soll es einer Drittanwendung möglich werden, beliebige Benutzungsschnittstellen für Dienste in einer Anwendung zu kombinieren und einheitliche Benutzungsschnittstellen automatisiert zu erzeugen, sowie erfasste Daten abschließend an konkrete Dienst-Implementierungen im Ökosystem weiterzureichen. Die Anforderungen an die Modellierung wurden in Abschnitt 3.1, Tabelle 3.1 unter **(A.4)** genannt. Sie bestehen in der Schaffung einer universellen Repräsentation (A4.1), einer Möglichkeit zur generischen Anbindung an Dienste eines Dienst-Ökosystems (A4.2) und der gemeinschaftlichen Nutzbarkeit in einer dezentralen Infrastruktur (A4.3).

Im Verlauf des Kapitels wurden die Ergebnisse der vorangegangenen Kapitel um Lösungen zur Erfüllung dieser Anforderungen erweitert. In Abschnitt 9.2 wurde eine **dezentrale Infrastruktur** vorgestellt, welche die gemeinsame Nutzung von Anwendungsmodellen als *Shared UIs* ermöglicht. In Abschnitt 9.4 wurde eine **universelle Beschreibung** für Anwendungsmodelle über Ontologien vorgestellt und in Abschnitt 9.5 eine Lösung zur **Anbindung von Diensten** im Dienst-Ökosystem über die Modellierung der Korrelation von Instanzdaten auf Dienst-Ontologie-Instanzen vorgeschlagen.

Das Ergebnis ist die Architektur für ein Benutzungsschnittstellen-Ökosystem, in welchem beliebige Anbieter Anwendungsmodelle und Verfahren für eine gemeinschaftliche Nutzung bereitstellen können. Das dargestellte Ökosystem schafft eine **Entkopplung der Auswahl, Generierung und Dienst-Anbindung** von konkreten Anbietern und ermöglicht damit die freie Wahl von Komponenten, die zur Erzeugung von Benutzungsschnittstelle kombiniert und wiederverwendet werden können.

Die vorgestellte Lösung geht damit über bestehende Ansätze hinaus bzw. fügt Lösungen hinzu, die dort bisher nicht adressiert wurden:

- Sie verwendet eine minimale Zahl universell beschriebener Artefakte als Basis zur vollständig automatisierten Generierung funktionsfähiger Komponenten. Diese sind kombinierbar und in verschiedenen fachlichen und technischen Kontexten nutzbar
- Sie verwendet gemeinschaftlich nutzbare funktionale Komponenten, die in einer dezentralen Infrastruktur von verschiedenen Anbietern verteilt angeboten werden können
- Sie ermöglicht die Anbindung an beliebige Dienst-Implementierungen in einem Dienst-Ökosystem

Der Ansatz unterscheidet sich von bestehenden Ansätzen insbesondere durch die Verwendung eines datenzentrierten Modells und dessen universeller Repräsentation. Die universelle Beschreibung über Ontologien ermöglicht die gemeinschaftliche Nutzung und Integration in multiple Umgebungen. Sowohl die Applikations-Ontologie als auch die Dienst-Ontologien sind frei verfügbares Wissen (*shared knowledge*) und sind damit jedem Teilnehmer in einem Ökosystem zugänglich.

10. Implementierung, Validierung und Evaluation

Inhalt dieses Kapitels ist der Nachweis, dass der in der Arbeit vorgestellte Ansatz eine Lösung der in den Abschnitten 1.2 und 1.3 formulierten Problemstellung und Zielsetzung darstellt. Hierzu wird für die entwickelten Artefakte die *Machbarkeit* nachgewiesen, deren *Anwendbarkeit* auf den Problembereich validiert und deren *Nutzbarkeit* anhand konkreter Anwendungsfälle evaluiert.

10.1 Zielsetzung und Vorgehen

Der **Forschungsgegenstand** der Arbeit ist ein Verfahren zur modellgetriebenen Entwicklung von Benutzungsschnittstellen für Dienste eines Dienst-Ökosystems. In Abschnitt 1.2 wurden die damit verbundenen Herausforderungen dargestellt sowie Eigenschaften aufgeführt, die ein Lösungsansatz aufweisen muss.

Die Lösung der Arbeit beruht auf der These, dass Benutzungsschnittstellen dialogabasierter Anwendungen in Struktur und Verhalten auf den Eigenschaften und der Semantik der verarbeiteten Daten beruhen. Daraus wurde abgeleitet, dass ein datenzentriertes Modell zu deren Beschreibung verwendet werden kann und zur vollständig automatisierten Erzeugung von Varianten und Kompositionen sowie zur gemeinschaftlich Nutzung in verteilten Dienst-Ökosystemen nutzbar ist. Diese durchgängige Lösung kann mit bestehenden Ansätzen nicht erreicht werden (vgl. Abschnitt 3.2) und ist die nachzuweisende **Hauptinnovation des entwickelten Ansatzes**.

Das Ziel dieses Kapitels liegt im Nachweis, dass die im Verlauf der Arbeit vorgestellten Artefakte (Konzepte, Modelle, Verfahren, Architektur) die in der Problemstellung geforderten Eigenschaften aufweisen und zur Lösung der Herausforderungen beitragen. Hierzu werden die Ergebnisse hinsichtlich ihrer Machbarkeit, Anwendbarkeit und Nutzbarkeit untersucht (vgl. Abschnitt 4.2). Die Untersuchungen zur **Machbarkeit** sollen belegen, dass die dargestellten Artefakte technisch umsetzbar sind. Die Untersuchungen zur **Anwendbarkeit** sollen belegen, dass sie den gestellten Anforderungen genügen und sich erwartungsgemäß verhalten (Validierung). Die Untersuchungen zur **Nutzbarkeit** sollen zeigen, dass die Artefakte in konkreten, nicht-künstlichen Anwendungsszenarien eingesetzt werden können und zu Verbesserungen bei der Erstellung von Benutzungsschnittstellen führen (Evaluation).

Untersuchte Artefakte und Bewertungskriterien

Zur Lösung der Problemstellung wurden drei technische Artefaktarten entwickelt. Insbesondere sind dies **Modelle** zur Beschreibung von Anwendungen, Varianten und Kompositionen, **Verfahren** zu deren Generierung sowie eine **Architektur** für deren Zusammenspiel in einem verteilten Kontext. Zur Modellierung und Generierung von Benutzungsschnittstellen sind dies konkret:

- **(AF.1)** Metamodell für Anwendungs- und Variantenmodell (Kapitel 6)
- **(AF.2)** Erweiterung des Metamodells für Kompositionen (Kapitel 7)
- **(AF.3)** Verfahren zur Generierung finaler Benutzungsschnittstellen (Kapitel 8)

Zur Umsetzung der gemeinschaftlichen Nutzung in einem Dienst-Ökosystem (Kapitel 9) werden folgende Artefakte hinzugefügt:

- **(AF.4)** Universelle, ontologische Repräsentation für Anwendungsmodelle
- **(AF.5)** Erweiterung des Metamodells zur Modellierung der Dienst-Anbindung
- **(AF.6)** Architektur für ein Benutzungsschnittstellen-Ökosystem für Shared UIs

In Kapitel 4, Abschnitt 4.2 wurde das methodische Vorgehen zur Evaluation der Arbeit vorgestellt und die Kriterien festgelegt, nach denen die Lösung im Verlauf des Kapitels bewertet wird. Die **Bewertungskriterien** ergeben sich direkt aus den in der Problemstellung (Abschnitt 1.2) geforderten Eigenschaften einer Lösung und sind hier nochmals zusammengefasst (vgl. Abschnitt 4.2):

- **(K.1)** Erzeugung von Benutzungsschnittstellen, die funktional manuell erstellten entsprechen
- **(K.2)** Erzeugung inhaltlicher Varianten
- **(K.3)** Komposition bestehender Benutzungsschnittstellen
- **(K.4)** Vollständig automatisierte Erzeugung technischer Varianten
- **(K.5)** Beschreibung gemeinschaftlich nutzbarer Benutzungsschnittstellen, die als Komponenten in Anwendungen Dritter integrierbar sind
- **(K.6)** Erzeugung von Benutzungsschnittstellen, die an beliebige Dienste eines Dienst-Ökosystems angebunden werden können

Angewendete Nachweismethoden

Zur Evaluation der Artefakte wurde *Design Science Research* (DSR) [52, 93] als methodischer Rahmen genutzt (s. Abschnitt 4.1). Abschnitt 4.2 untersucht Evaluationsmethoden in DSR und legt das methodische Vorgehen im Rahmen dieser Arbeit fest. Hierbei wird auf verbreitete Methoden zur Evaluation verschiedener Artefakttypen zurückgegriffen und deren Anwendbarkeit auf die erstellten Artefakte (AF.1)-(AF.6) untersucht. Insbesondere wird dargelegt, welche Methoden sich zum Nachweis der Machbarkeit, Anwendbarkeit und Nutzbarkeit sowie zur Bewertung der Kriterien (K.1)-(K.6) eignen (vgl. Abschnitt 4.2, Tabelle 4.3).

Der Nachweis der **Machbarkeit** erfolgt über die *prototypische Umsetzung* der Artefakte. Für jedes Artefakt wird eine Implementierung erstellt, welche die in den Kapiteln 6 bis 9 beschriebenen Modelle, Verfahren und Architekturkomponenten technisch umsetzt. Die **Anwendbarkeit** der Artefakte wird über deren praktische Nutzung zur Beschreibung und Generierung von Anwendungsvarianten validiert. Hierzu werden *künstliche Szenarien* verwendet, an denen die Erfüllung der Anforderungen an den Lösungsansatz gezeigt werden. Zudem werden ggf. existierende Benutzungsschnittstellen zum Vergleich herangezogen. Die Untersuchung zur **Nutzbarkeit** des Ansatzes erfolgt sowohl als artefaktübergreifende Evaluation in Form realitätsnaher Fallstudien (*Case Studies*), als auch durch den Einsatz von Ergebnissen in einem realen Umfeld (*Action Research*).

Tabelle 10.1 stellt die in den folgenden Abschnitten durchgeführten Untersuchungen im Überblick zusammen. Die Abschnitte sind nach der Art der Nachweise (*Machbarkeit, Anwendbarkeit, Nutzbarkeit*) gegliedert. Für jeden Abschnitt ist vermerkt, welche Nachweismethode angewendet wurde, welche Bewertungskriterien und welche Artefakte im Fokus stehen. Die letzte Spalte beschreibt das Ziel des Nachweises und wie dieser erbracht wird.

Abschnitt		Typ	Krit.	Artefakte	Nachweis
10.2 Machbarkeit (Implementierung)					
10.2	**Implementierung der Artefakte**	Prototype	./.	Alle	Machbarkeit durch Implementierung aller Artefakte des Ansatzes
10.3 Anwendbarkeit (Validierung)					
10.3.1	**Datenzentrierte Modellierung von Benutzungsschnittstellen**	Scenario	K.1 K.2 K.3	AF.1 AF.2	Anwendbarkeit der datenzentrierten Modellierung von Anwendungsvarianten, Kompositionen über künstliche Szenarien und Vergleich mit existierenden Anwendungen
10.3.2	**Verfahren zur Generierung technischer Varianten**	Scenario	K.4	AF.3	Anwendbarkeit zur automatischen Herleitung von Varianten für konkrete Interaktions- und Technologiekontexte (mobile, web, desktop)
10.3.3	**Universelle Beschreibung von SharedUIs über Ontologien**	Scenario	K.5	AF.4	Anwendbarkeit der ontologischen Repräsentation über Vergleich der Mächtigkeit zu mimesis.DSL
10.4 Nutzbarkeit (Evaluation)					
10.4.1	**Produktiver Einsatz in *Anwendungen zur Risikoanalyse***	Action Reserach	K.4	AF.3	Nutzbarkeit des Verfahrens zur Herleitung technischer Varianten durch Einsatz in einem realen Szenario
10.4.2	**Demonstrator: Freie Kombination in einem *Dienstleistungs-Selektor***	Case Study	K.5 K.6	AF.4 AF.5 AF.6	Nutzbarkeit zur Verteilung und Integration in Drittanwendungen in einem realitätsnahen, künstlichen Szenario
10.4.3	**Nutzung des Ansatzes zum Aufbau eines verteilten Marktplatzes (*Distributed Marketspace)***	Action Research	K.5 K.6	AF.4 AF.5 AF.6	Nutzbarkeit der Verteilung, Integration und Datentransformation, sowie Anbindung von Diensten in einem realen Szenario

Tabelle 10.1. Durchführung der Untersuchungen

10.2 Implementierung der Artefakte

Im Folgenden wird die technische Umsetzung der Artefakte (AF.1)-(AF.6) als Komponenten dargestellt, die in den folgenden Abschnitten zur Validierung und Evaluation sowie zum Aufbau einer Infrastruktur für ein Benutzungsschnittstellen-Ökosystem genutzt werden. In Anhang A.7.4 werden ergänzend Verweise auf die online zur Verfügung gestellte Dokumentation der implementierten Dienste angegeben.

10.2.1 Implementierung der Modelle, Notationen und Verfahren

Abbildung 10.1 zeigt die erstellten Komponenten zur Umsetzung der Artefakte des Anwendungsmodells, der Variantenbeschreibung und des Generierungsverfahrens (AF.1)-(AF.4). Die zentralen Komponenten sind Implementierungen des **mimesis Metamodells** und der **mimesis Variantenbeschreibung**, welche zur Laufzeit dem Generierungsverfahren die Informationen von Anwendungsbeschreibungen und Varianten bereitstellen. Die Implementierung erfolgt in der Programmiersprache Java als Klassen, mit denen zur Laufzeit aus Anwendungsbeschreibungen Instanzen der Metamodelle erzeugt werden.

Als **Beschreibungssprachen** zur Repräsentation des Anwendungsmodells wurde die in Anhang A.3.1 spezifizierte **mimesis.DSL** sowie die universelle Repräsentation als Ontologie **mimesis.rdfowl** umgesetzt (Abschnitt 9.4). Zusätzlich wurde eine JSON- und XMLSchemabasierte Notation (vgl. [147]) prototypisch entwickelt (**mimesis.json**, **mimesis.xml**). Für alle Sprachen wurden **Im-/Export-Module** erstellt, welche die Repräsentation zur Laufzeit in eine *mimesis Metamodell-Instanz* überführen.

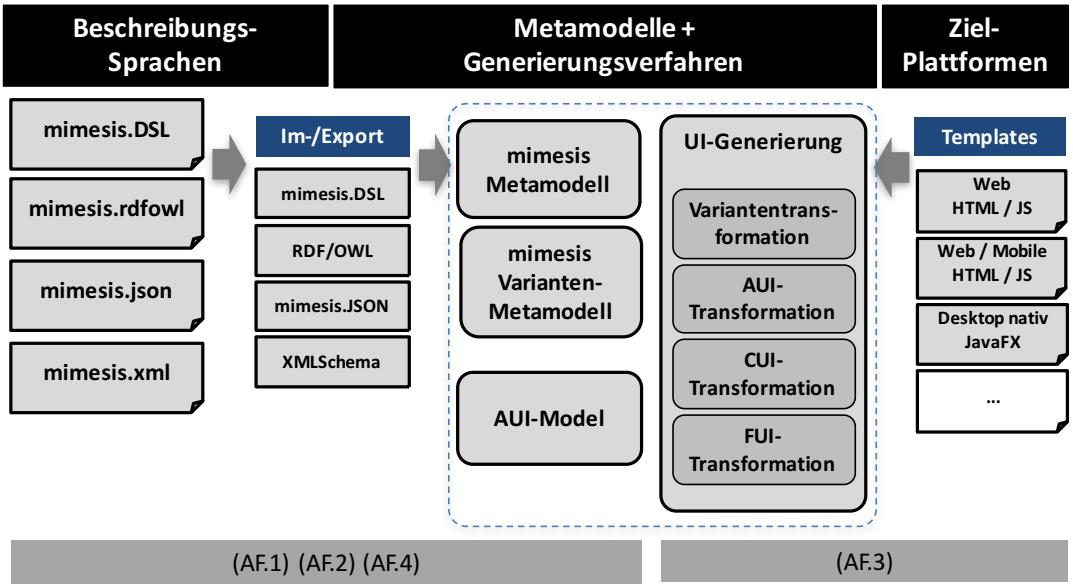


Abbildung 10.1. Komponenten zur Umsetzung der Artefakte (AF.1)-(AF.4)

Zur Umsetzung des **Generierungsverfahrens** wurden die in Kapitel 8 dargestellten Transformationsschritte als Java-Komponente umgesetzt. Hierzu wurde das **AUI-Modell** (vgl. Abschnitt 8.4.1) implementiert, welches zur abstrakten Beschreibung der Benutzungsschnittstelle während der AUI- und CUI-Transformation verwendet wird (vgl. Abschnitt 8.4.2 und 8.4.3). Das Generierungsverfahren erhält als Eingabe eine *mimesis Metamodell-Instanz* und optional eine *mimesis Variantenbeschreibung*. Es führt die in Abschnitt 8.2 dargestellten Transformationen durch. Zur *FUI-Transformation* wurde dabei der in Abschnitt 8.4.4 beschriebene templatebasierte Ansatz umgesetzt und für ausgewählte Zielplattformen (*web*, *web/mobile*, *desktop*) entsprechende Templates erstellt.

Erweiterungen zur Umsetzung von *SharedUIs*. Zur Umsetzung der **Dienstanbindung** und der **Instanzdatentransformation** erfolgte eine Ergänzung der o.g Komponenten. Abbildung 10.2 stellt die Ergänzungen dar. Zur Modellierung der Dienstanbindung (AF.5) wurde eine Erweiterung des Metamodells und der Beschreibungssprachen um die erforderlichen Korrelationsinformationen vorgenommen (vgl. Abschnitt 9.5).

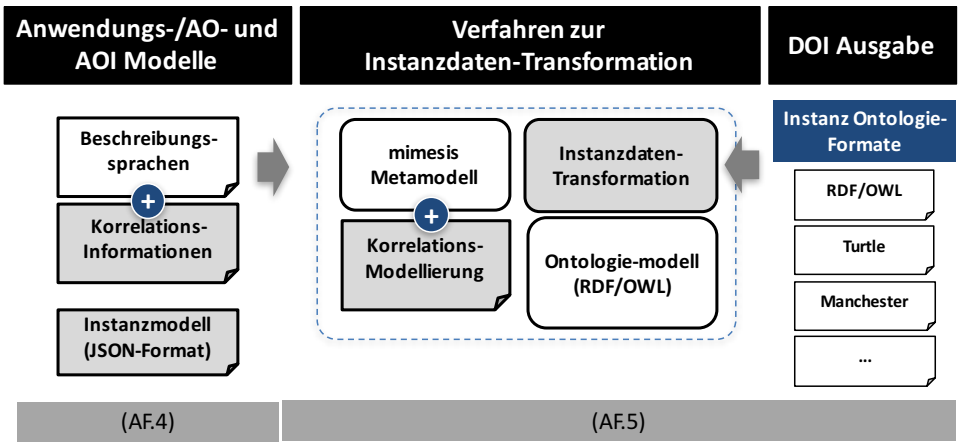


Abbildung 10.2. Komponenten zur Umsetzung der Artefakte (AF.4)-(AF.5)

Zudem wurde ein **Modell für Instanzdaten** im JSON-Format zur Repräsentation der zur Laufzeit erfassten Daten erstellt. Die Implementierung der **Instanzdatentransformation** verwendet das erweiterte Metamodell und das JSON-Instanzdatenmodell zur Erzeugung einer Instanz-Ontologie zur Laufzeit. Zur Implementierung der Transformation wurde eine existierende Java-Bibliothek zur Arbeit mit Ontologien genutzt (OWL-API [161]), welche den Export des Instanz-Modells in unterschiedliche Repräsentationen ermöglicht (z.B. *RDF/OWL*, *Turtle*, *Manchester*).

10.2.2 Infrastruktur des Benutzungsschnittstellen-Ökosystems

Zur Evaluation der vorgeschlagenen Architektur des Benutzungsschnittstellen-Ökosystems wurden die in Abschnitt 9.2 dargestellten Infrastrukturkomponenten (Abbildung 9.2, *Anwendungsmodell-Repository* Ⓐ, *UI-Dienst* Ⓑ und *Instanzdaten-Dienst* Ⓒ) als *RESTful Webservices* in der Programmiersprache Java implementiert. In Abbildung 10.3 sind die erstellten Dienst-Implementierungen dargestellt. Zur Umsetzung wurden die in Abschnitt 10.2.1 erstellten Implementierungen genutzt.

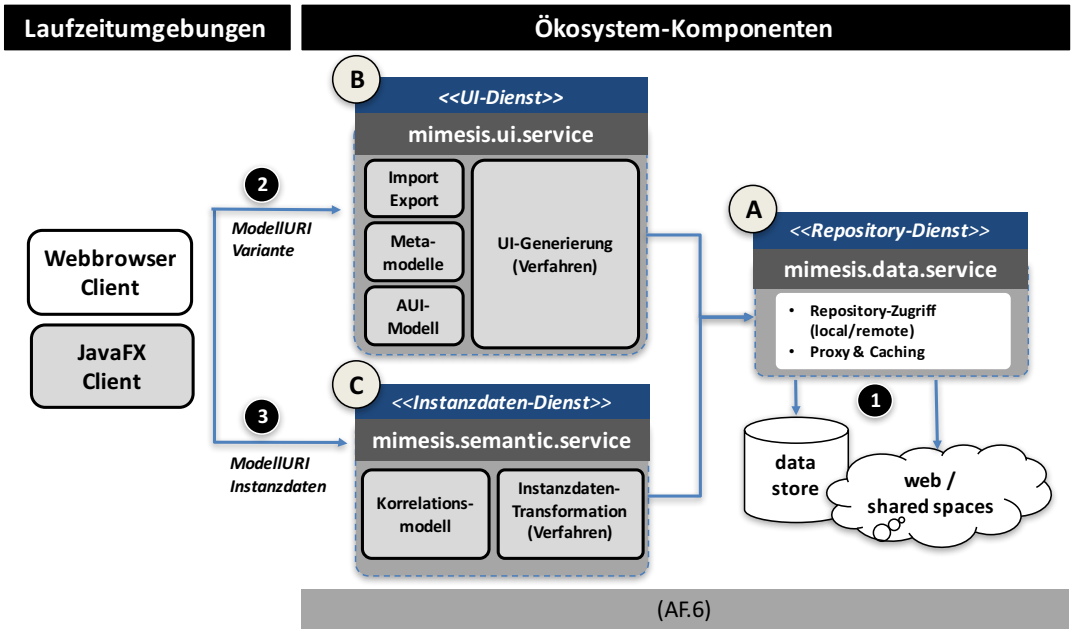


Abbildung 10.3. Umsetzung der Infrastrukturkomponenten (AF.6)

Anwendungsmodell-Repository Ⓐ: Zum Zugriff auf Anwendungsmodelle dient der *mimesis.data.service*, welcher Modelle, Variantenbeschreibungen und Assets entweder aus einer angebundenen Datenhaltung (*data store*) oder aus entfernten Quellen (*web /shared space*) anhand ihrer URI bezieht ①. Der Dienst übernimmt die Aufgabe eines *Proxys* zum Zugriff auf Ressourcen sowie das Zwischenspeichern (*caching*) von Anfrageergebnissen. Dieser Service ist die zentrale Quelle zum Bezug von Artefakten für die anderen Infrastrukturkomponenten.

UI-Dienst Ⓑ: Die Rolle des UI-Dienstes übernimmt der *mimesis.ui.service*. Er implementiert die Generierung der finalen UI für unterschiedliche Zielplattformen. Als Eingabe wird die Referenz auf ein Anwendungsmodell und die zu erzeugende Variante ② erwartet (als URI bzw. Schlüssel im *data-store*). Diese werden aus dem *Anwendungsmodell-Repository* bezo-

gen und in eine FUI transformiert. Zusätzlich bietet der Dienst Schnittstellen zur Inspektion von Zwischenergebnissen, die während der Transformation entstehen (z.B. Komposition, AUI-Modell).

Instanzdaten-Dienst ©: Die Aufbereitung der Instanzdaten zur Dienstanbindung ist im *mimesis.semantic.service* implementiert. Die prototypischen Umsetzung erzeugt *Instanz-Ontologien*, die aus den Korrelationsinformationen im Anwendungsmodell hergeleitet werden. Als Eingabe erhält der Dienst die Referenz auf ein Anwendungsmodell (als URI bzw. Schlüssel im *data-store*) sowie die eingegebenen Daten der Benutzungsschnittstelle im JSON-Format ③ und liefert die entsprechende Instanz-Ontologie.

Die Ausführung der webbasierten Benutzungsschnittstellen erfolgt in dieser Infrastruktur in **HTML-basierten Webbrowsern**, sodass zur Umsetzung dieser Laufzeitumgebung keine Implementierung erforderlich ist. Zusätzlich wurde prototypisch eine generische **JavaFX-Anwendung** erstellt, die zur Durchführung des Nachweises der Anwendbarkeit für Desktop-Anwendungen verwendet wird.

10.3 Validierung der Artefakte

Die Validierung soll zeigen, dass die erstellten Artefakte (AF.1-AF.6) den Anforderungen (vgl. Abschnitt 3.1) genügen. Die Untersuchungen werden im Folgenden in drei Bereiche gegliedert:

- Datenzentrierte Modellierung von Benutzungsschnittstellen (Abschnitt 10.3.1)
- Generierungsverfahren zur Erzeugung technischer Varianten (Abschnitt 10.3.2)
- Universelle Beschreibung über Ontologien für SharedUIs (Abschnitt 10.3.3)

In Abschnitt 10.3.1 wird validiert, dass der datenzentrierte Ansatz und die vorgeschlagenen Artefakte zur Modellierung dialogbasierter Benutzungsschnittstellenvarianten und Kompositionen geeignet sind (K.1-K.3). In Abschnitt 10.3.2 wird die Eignung des Ansatzes zur Generierung technischer Varianten (K.4) gezeigt. Abschnitt 10.3.3 zeigt die Anwendbarkeit der ontologischen Beschreibung zur Modellierung von Benutzungsschnittstellen (K.5). Für jeden Bereich werden zuerst die Ziele, Kriterien und Methodik für den Nachweis angegeben. Anschließend wird der Versuchsaufbau und die Durchführung dargestellt, gefolgt von den Ergebnissen.

In Anhang A.7.2 finden sich Verweise zu Artefakten und Prototypen, an denen die Anwendungsfälle nachvollzogen werden können.

10.3.1 Datenzentrierte Modellierung von Benutzungsschnittstellen

Zielsetzung, Kriterien und Methodik. Das Ziel der Validierung des datenzentrierten Ansatzes liegt im Nachweis, dass

- voll funktionsfähige Anwendungen aus den Informationen des Metamodells ableitbar sind und dass die in der Analyse identifizierten Interaktionsmuster umgesetzt werden können **(K.1)**

- mit der Modellierung voll funktionsfähige inhaltliche Varianten redundanzfrei beschrieben und erzeugt werden können **(K.2)**
- bestehende Anwendungsmodelle kombiniert und Komponenten innerhalb der Komposition angepasst werden können **(K.3)**

Als Evaluierungsmethode wurde die Umsetzung realitätsnaher Szenarien gewählt (vgl. Tabelle 10.1). In Zusammenarbeit mit der Allianz Deutschland AG wurden hierzu die in der Analyse herangezogenen Anwendungen und Varianten aus dem Versicherungsbereich modelliert und umgesetzt. Ergänzend wurden Beispiele aus anderen Domänen modelliert (z.B. Flugbuchung). Als Szenarien im Versicherungsbereich wurden repräsentative Antragstrecken nachgebildet (z.B. *Haftpflichtversicherung*, *Reiseversicherung*, *Schadenmeldung*). Hinzu kamen *Kontaktfomulare* für diverse Anwendungsfälle (z.B. *Adressänderung*, *Familienstand* etc.). Als inhaltliche Varianten wurden Beispiele für *Vertreter* und *Endkunden* umgesetzt. Als (künstliches) Szenario für Kompositionen wurde die Buchung einer Reise mit kombinierbaren Produktkomponenten (*Flugbuchung*, *Mietwagen*, *Reiseversicherung* etc.) umgesetzt, welche Anwendungsmodelle aus den vorigen Evaluationsschritten kombiniert.

Durchführung. Abbildung 10.4 zeigt den Aufbau der Umgebung zur Validierung. Zum Nachweis von (K.1) wurden im ersten Schritt für die ausgewählten Szenarien Anwendungsmodelle erstellt ①. Für den Nachweis von (K.2) wurden diese um Variantenbeschreibungen ergänzt. Für (K.3) wurden diese als Komponenten in Kompositionen wiederverwendet und über Variantenbeschreibungen modifiziert. Zur Beschreibung der Modelle wurde *mimesis.DSL* als Notation genutzt.

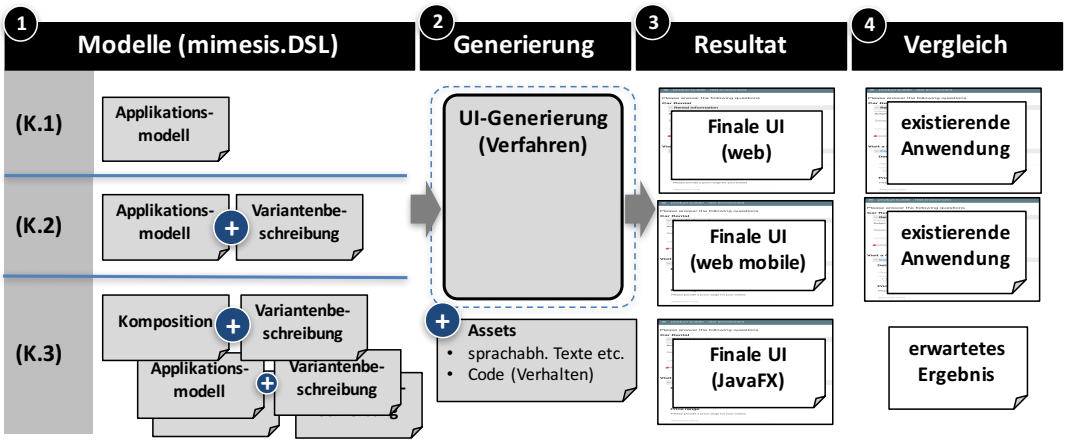


Abbildung 10.4. Aufbau der Umgebung zur Validierung (Modellierung)

Ergänzend wurden die zur Generierung der finalen Benutzungsschnittstelle benötigten ***Assets*** erstellt (② unten). Insbesondere waren dies sprachabhängige Texte z.B. für Labels und Icons, sowie der Sourcecode zur Implementierung des Verhaltens der Anwendung. Die Implementierung wurde in Form von wiederverwendbaren JavaScript-Modulen umgesetzt, die für unterschiedliche Technologiekontexte genutzt werden konnten.

Aus diesen Artefakten wurden mittels der Implementierung des Generierungsverfahrens ② FUIs hergeleitet ③, die hinsichtlich ihrer Funktionalität und der Anforderungen geprüft wurden. Die Validierung erfolgte einerseits durch Vergleich mit den an die Lösung gestellten

Anforderungen aus Abschnitt 3.1, Tabelle 3.1, andererseits mit bestehenden Anwendungen, die als Grundlage für die Erstellung des Metamodells dienten ④.

Ergebnisse. Mittels der dargestellten Vorgehensweise konnte die Anwendbarkeit des datenzentrierten Ansatzes nachgewiesen werden. Es wurde demonstriert, dass die grundsätzliche Modellierung über ein einziges, intuitiv erfassbares Artefakt mit geringer Komplexität erfolgen kann (vgl. Beispiele in Anhang A.7.2). Die im vorgeschlagenen Metamodell enthaltenen Informationen genügten zur Herleitung nicht-trivialer Benutzungsschnittstellen, die den Anforderungen hinsichtlich Struktur, typisierter Eingabe und Verhalten entsprechen. Für alle ausgewählten Szenarien konnten lauffähige Ergebnisse erzeugt werden, die funktional den manuell erstellten Gegenstücken entsprachen. Inhaltliche Varianten konnten über ein weiteres Artefakt redundanzfrei modelliert, sowie Kompositionen erstellt und vollständig automatisiert generiert werden. Die darüber hinaus benötigten *Assets* (Texte, Sourcecode zur Umsetzung des Verhaltens) konnten variantenspezifisch erstellt und somit die vollständige Generierung von Anwendungsvarianten und Kompositionen gezeigt werden.

Erwartungsgemäß ergaben sich die Einschränkungen, die bereits in der Diskussion in den Abschnitten 6.6 und 7.5 angeführt wurden. So besteht eine Einschränkung bei Fällen, in denen die Reihenfolge der Fragen in Varianten variiert. Dies kann jedoch zukünftig durch Erweiterung der Variantendefinition gelöst werden.

10.3.2 Generierungsverfahren zur Erzeugung technischer Varianten

Zielsetzung, Kriterien und Methodik. Das Ziel liegt im Nachweis, dass mittels des entwickelten Modellierungsansatzes und des Generierungsverfahrens **(AF.3)** technische Varianten automatisch generierbar sind. Hierzu soll demonstriert werden, dass lauffähige Anwendungen erzeugt werden, die hinsichtlich ihrer Struktur, Interaktionselementen und dem Verhalten an unterschiedliche Interaktions- und Technologiekontexte angepasst sind **(K.4)**. Es sollte validiert werden, dass die im Metamodell enthaltenen Informationen zur Generierung qualitativ hochwertiger Benutzungsschnittstellenvarianten genügen und auf unterschiedliche Kontexte anwendbar sind. Ein weiteres Ziel ist der Nachweis, dass semantische Annotationen zur Erzeugung spezifischer Interaktionselemente nutzbar sind.

Als Nachweismethode wurde die Umsetzung realitätsnaher Szenarien gewählt (vgl. Tabelle 10.1). Hierbei dienten die Anwendungsmodelle und Variantenbeschreibungen aus den vorangegangenen Evaluierungen als Basis. Als Technologie- und Interaktionskontexte wurden folgende Repräsentanten gewählt:

- webbasierte Browseranwendungen
- mobile webbasierte Anwendungen
- Java Anwendungen (Rich-Client)

Durchführung. Abbildung 10.5 zeigt den Aufbau der Umgebung zur Validierung. Hierzu wurden die Modelle und Assets der vorangegangenen Evaluierungen (Modellbasis ①) als Szenarien wiederverwendet und als Eingabe für die Generierung ③ genutzt. Mittels der Varianten-, AUI-, CUI- und FUI-Transformation wurde daraus eine FUI erzeugt. Hierbei wurde das in Abschnitt 8.4.4 dargestellte Template-basierte Verfahren für deklarative Beschreibungssprachen umgesetzt und entsprechende Template-Module für webbasierte (HTML/JavaScript) und JavaFX (FXML/JavaScript) Anwendungen erstellt ②.

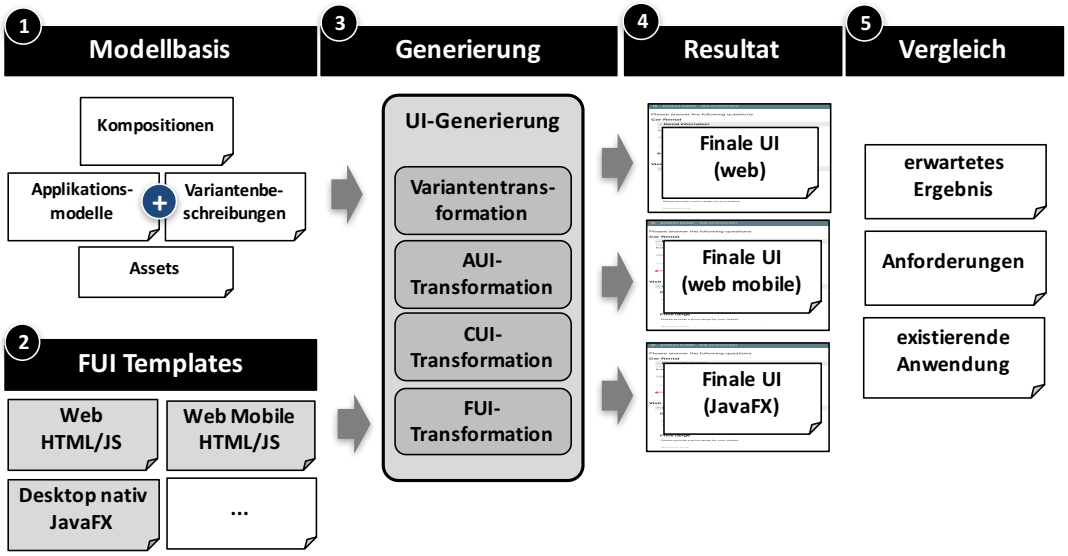


Abbildung 10.5. Aufbau der Umgebung zur Validierung (Generierung)

Die generierten Resultate ④ wurden anschließend bezüglich der Anforderungen (vgl. Tabellen 5.1-5.4, Abschnitte *Technische Varianten*) validiert. Sofern vergleichbare Anwendungen aus der Analyse verfügbar waren, wurden die generierten Benutzungsschnittstellen mit den Vorlagen verglichen. Zur Validierung darüber hinausgehender Eigenschaften wurden die erwarteten Ergebnisse validiert, z.B. bei der Verwendung semantischer Annotationen für spezifische Interaktionselemente ⑤.

In Anhang A.7.2 finden sich Informationen und Ergebnisse zu ausgewählten Szenarien sowie Verweise zu Artefakten und Prototypen, an denen Anwendungsfälle nachvollzogen werden können.

Ergebnisse. Mit dieser Umsetzung konnte nachgewiesen werden, dass aus der datenzentrierten Beschreibung inhaltliche Varianten und Kompositionen einer Benutzungsschnittstelle abgeleitet und daraus schrittweise lauffähige Anwendungen für multiple Interaktions- und Technologiekontexte erzeugt werden können.

Die generierten Resultate entsprachen dabei funktional den Anforderungen (vgl. Tabellen 5.1-5.4, Abschnitte *Technische Varianten*). Sie konnten die im Metamodell enthaltenen Interaktionsmuster für verschiedene Interaktions- und Technologiekontexte umsetzen. Mittels semantischer Annotationen konnten variierende Interaktionselemente erzeugt werden, wodurch die Generierung nicht-trivialer, nutzergruppenspezifischer Anwendungen demonstriert werden konnte.

Einschränkungen konnten beobachtet werden, wenn zielplattformspezifische Interaktionsformen über die im Metamodell vorgesehenen Informationen hinausgingen (z.B. die Reaktion auf Gesten auf mobilen Geräten). Insbesondere bei der Webanwendungsentwicklung ist dies ein in der Praxis bekanntes aber noch nicht gelöstes Problem. Dies könnte zukünftig im Anwendungsmodell durch Modellierung plattformspezifischer Zusätze gelöst werden, was jedoch die Nutzbarkeit der Modelle in beliebigen Kontexten einschränken würde.

10.3.3 Universelle Beschreibung von SharedUIs über Ontologien

Zielsetzung, Kriterien und Methodik. Das Ziel liegt im Nachweis, dass die vorgeschlagene ontologische Repräsentation zur Beschreibung von Benutzungsschnittstellen angewendet werden kann. Dadurch soll eine universelle und austauschbare Beschreibung der im Metamodell enthaltenen Informationen geschaffen werden. Dies trägt wiederum zur Erfüllung des Kriteriums **(K.5)** bei, welches eine gemeinschaftlich nutzbare Beschreibung fordert.

Hierzu wird nachgewiesen, dass die ontologische Repräsentation dieselbe Mächtigkeit hinsichtlich der enthaltenen Informationen besitzt wie das Metamodell bzw. die *mimesis.DSL*. Der Nachweis erfolgte über eine vergleichende Validierung (vgl. Tabelle 10.1). Dabei wurden für Szenarien der vorangegangenen Evaluierungen ontologische Beschreibungen der Benutzungsschnittstellen erstellt. Daraus wurden Instanzen des Metamodells erzeugt, welche mit den entsprechenden Instanzmodellen der korrespondierenden *mimesis.DSL* inhaltlich verglichen wurden.

Durchführung. Abbildung 10.6 zeigt den Versuchsaufbau und die Schritte der Validierung. Für bestehende *mimesis.DSL*-Beschreibungen wurde je eine entsprechende ontologische Repräsentation in *mimesis.rdfowl* erstellt ①. Daraus wurde über den **Import** (vgl. Abschnitt 10.2.1) ② eine Metamodell-Instanz erzeugt. Analog wurde die korrespondierende *mimesis.DSL*-Beschreibung importiert. Die resultierenden Metamodell-Instanzen wurden dann auf Äquivalenz geprüft ③.

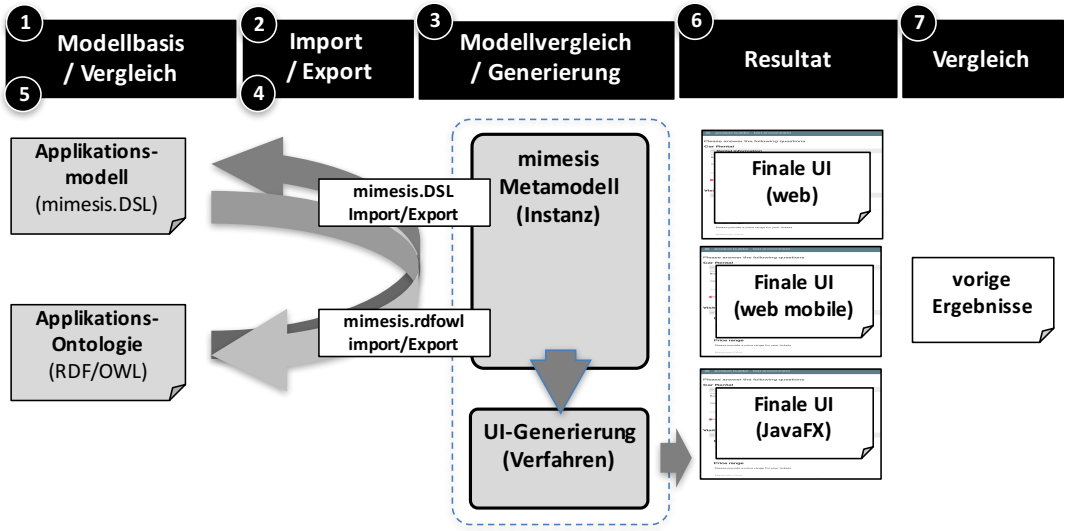


Abbildung 10.6. Aufbau der Umgebung zur Validierung (universelle Repräsentation)

Ein weiterer Schritt bestand im **Export** eines importierten Modells in das jeweils andere Format ④. Auch diese Ergebnisse wurden auf Äquivalenz verglichen ⑤. Anschließend wurden die Schritte ①-⑤ mit den Ergebnismodellen aus Schritt ⑤ der vorigen Iteration erneut durchgeführt. So wurde sichergestellt, dass sowohl Quell- als auch Zielmodell dieselben Informationen enthalten und die Überführung in die jeweils andere Repräsentation verlustfrei erfolgen kann.

Zur Vervollständigung der Prüfungen wurden für alle ontologischen Repräsentationen FUIs erzeugt ⑥ und mit den generierten Benutzungsschnittstellen aus den DSL-Beschreibungen auf Übereinstimmung verglichen ⑦.

Ergebnisse. Mittels des Versuchsaufbaus konnte demonstriert werden, dass die ontologische Repräsentation die gleiche Mächtigkeit hinsichtlich der Metamodellinhalte aufweist wie die *mimesis.DSL*[50]. Beide Repräsentationen konnten auf gleiche Metamodell-Instanzen abgebildet und ineinander überführt werden. Die Generierung der FUI führte erwartungsgemäß zu den selben Ergebnissen.

Die Evaluierung zeigte, dass mit der *mimesis.DSL* (als proprietäre Beschreibung) Modelle manuell signifikant einfacher und fehlerfreier zu erstellen waren als dies mit der ontologischen Repräsentation möglich war. Die manuelle Erstellung umfangreicher Ontologien erwies sich als wenig intuitiv und fehleranfällig. Nachdem die Gleichmächtigkeit an Beispielen nachgewiesen werden konnte, wurde daher für weitere Untersuchungen dazu übergegangen, die Modelle mit *mimesis.DSL* zu erstellen und in *mimesis.rdfowl* zur konvertieren. Diese Erkenntnis **beeinträchtigt nicht die Anwendbarkeit des Ontologie-Ansatzes** in der Praxis. Aufgrund der Gleichmächtigkeit können Modelle als DSL erstellt und automatisiert in eine gemeinschaftlich nutzbare Repräsentation überführt werden.

10.4 Evaluation in realen Szenarien

Dieser Abschnitt stellt die durchgeführte Evaluation zur Nutzbarkeit des in der Arbeit vorgeschlagenen Ansatzes in einem realen bzw. realitätsnahen Umfeld dar. Hierzu wurden als Nachweismethoden Fallstudien (sog. *Case Studies*) und der Einsatz in einem bereits existierendem Umfeld (sog. *Action Research*) genutzt. Während in *Case Studies* ein *realitätsnahes* Szenario aufgebaut wird, das ggf. Vereinfachungen zugunsten der Nachvollziehbarkeit zulässt, wird beim *Action Research* ein *reales Szenario* vollständig gemäß der dort geltenden Anforderungen umgesetzt. Existiert hier bereits eine Umsetzung, wird zudem gezeigt, dass die gefundene die bisherige Lösung verbessert.

Die Nachweise erfolgten am Beispiel folgender Szenarien, die in den weiteren Abschnitten dargestellt werden:

- Action Research: Produktiver Einsatz in *Anwendungen zur Risikoanalyse* (Abschnitt 10.4.1)
- Case Study: Freie Kombination von Dienstleistungen in einem *Dienstleistungs-Selektor* (Abschnitt 10.4.1)
- Action Research: shareable UIs in Linked Data Dienst-Ökosystemen am Beispiel von Distributed Marketspaces (Abschnitt 10.4.3)

Einleitend wird der Kontext des Szenarios skizziert gefolgt von einer Beschreibung des Evaluations-Ziels. Anschließend wird der Versuchsaufbau und die Durchführung beschrieben, gefolgt von den Ergebnissen der Untersuchung.

[50] Dies bedeutet nicht, dass die Mächtigkeit der DSL gleich der von Ontologien ist. Es ist zwar möglich, die Inhalte der DSL vollständig auf OWL-Konstrukte abzubilden, nicht jedoch eine beliebige Ontologie als *mimesis.DSL* darzustellen. Es handelt sich hierbei um eine injektive Abbildung, in deren Rahmen eine Rücktransformation möglich ist.

10.4.1 Action Research: Produktiver Einsatz in *Anwendungen zur Risikoanalyse*

Das Ziel dieser Evaluierung besteht im Nachweis der vollständigen Generierbarkeit nichttrivialer, technischer Benutzungsschnittstellenvarianten mittels des vorgeschlagenen Generierungsansatzes (K.4). Als Nachweismethode wird *Action Research* verwendet, welches die Nutzbarkeit des Generierungsverfahrens in einem bestehenden Umfeld zeigt.

Kontext: Risikofragen in Anwendungen der Allianz Deutschland AG

Viele Produkt-Antragstrecken der Allianz Deutschland AG benötigen eine komplexe Risikoanalyse im Vorfeld des Abschlusses, die sich auf die Berechnung des Produkttarifs auswirkt. Hierzu werden abhängig vom Produkt zu verschiedenen Themenbereichen (z.B. Gesundheit, Berufsspezifika, Auslandsaufenthalte) Fragen gestellt, die ggf. weitreichende Nachfragen erfordern (z.B. Medikation und Verlauf einer angegebenen Erkrankung). Die Antragstrecken existieren in inhaltlichen Varianten für diverse Nutzergruppen (z.B. Vertrieb, Makler, Endkunden) und werden auf unterschiedlichen Plattformen bereitgestellt (z.B. Intranet, Maklerportale, Internetportale). Zur Bestimmung der Fragefolgen wird in Antragstrecken ein Risikoanalyse-Dienst verwendet, welcher die möglichen Fragedialoge modelliert. Dieser entscheidet über die zu erfassenden Daten für einen Anwendungsfall und bestimmt die zu erfassenden Informationen zu einem Zeitpunkt, basierend auf bereits erfolgten Eingaben.

Zielsetzung und Kriterien. Das Ziel der Evaluierung besteht darin, generisch erzeugte Risikofragedialoge in Antragstrecken einzubinden. Die Lösung muss folgende Kriterien erfüllen: (1) Eine Antragstrecke soll bestimmen, welcher Anwendungsfall für Risikofragen ausgeführt werden soll. (2) Dessen Benutzungsschnittstelle soll automatisch erzeugt und als Teil der GUI im Sinne eines Moduls als selbständig laufende Einheit nahtlos im Fragefluss der Antragstrecke integriert werden. (3) Eingaben sollen dem Risikoanalyse-Dienst gemeldet und Inhalte, Darstellung und Navigation in jedem Schritt dynamisch angepasst werden. Nach Beantworten der Risikofragen soll (4) das Ergebnis von der Antragstrecke weiterverarbeitet werden. Der besondere Fokus dieser Evaluierung liegt auf der *Erzeugung nicht-trivialer Benutzungsschnittstellenvarianten*. Insbesondere ist nachzuweisen, dass technische Varianten erzeugt werden können, die eine Vielfalt von Interaktionsformen unterstützen und spezifische Interaktionselemente erzeugen können, die aufgrund der Semantik der Daten bestimmt werden.

Die **Verbesserung**, die durch Einsatz des Artefakts erreicht werden soll, ist die generische Anbindung von beliebigen Prozessen zur Risikodatenerfassung und deren automatische Darstellung in Antragstrecken auf unterschiedlichen Plattformen.

Durchführung. Abbildung 10.7 zeigt den Aufbau und die Lösungsarchitektur für die Evaluierung. Ⓐ stellt die Antragstrecken schematisch dar, in deren Fragefluss die Risikofragen zu integrieren sind. In der Umsetzung wurden Rahmenanwendungen als *Java-Portlets bzw. Java Server Pages* (Vertreter-/Maklerportal) sowie *HTML-basierte Web-Anwendungen* (Antragstrecken im Kundenportal) verwendet. Ⓑ stellt den vorhandenen Allianz Riskanalysis Service (**ARS**[51]) dar, der den Fragedialog steuert.

Als zentrale Komponente für die Erzeugung der Benutzungsschnittstelle für die Risikofragen-Dialoge wurde ein *RESTful*-Webservice erstellt Ⓒ. Dieser wird von der

[51] Der Name des Services wurde für die Darstellung in der Arbeit geändert.

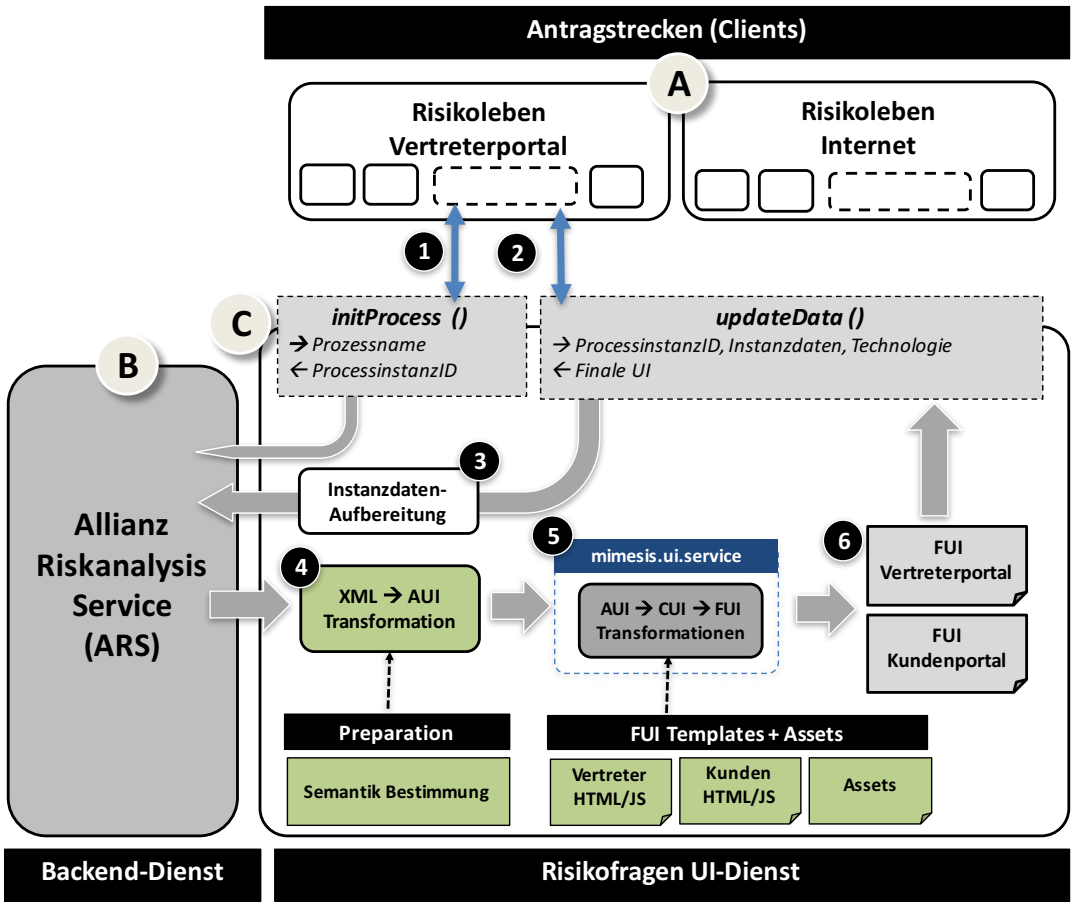


Abbildung 10.7. Generierung der Risikofrage-Dialoge

Antragstrecke zur Bereitstellung der UI für einen Frageverlauf (*Prozess*) aufgerufen und kommuniziert hierfür mit dem ARS. Er bietet eine Schnittstelle zum Start eines Prozesses (*initProcess()*) sowie zur Übermittlung von Eingaben an den ARS (*updateData()*). Das Ergebnis ist eine FUI zur Integration in die Seite der Antragstrecke.

Im ersten Schritt ① wird der Fragedialog initialisiert, indem der Prozess (*Prozessname*) an Ⓒ übermittelt wird. Die Rückgabe ist eine *ProzessinstanzID*, die zur Kommunikation mit dem ARS benötigt wird. Anschließend wird die FUI für eine spezifische *Technologie* angefordert ②. Dieser Aufruf erfolgt bei jeder weiteren Nutzereingabe. Hierbei werden die übermittelten *Eingabedaten* für die *ProzessinstanzID* aufbereitet ③ und dem ARS übergeben. Dieser ermittelt die weiteren zu erfassenden Daten und sendet diese in einer proprietären XML-Beschreibung als Antwort, welche in ein **mimesis AUI-Modell** überführt wird ④. Das vom ARS verwendete Modell **entspricht einer Teilmenge des mimesis Metamodells**, beinhaltet jedoch keine Informationen zur Semantik der erfassten Daten. Die fehlenden Informationen werden in diesem Schritt angereichert, sodass die Generierung spezifischer Interaktionselemente möglich ist. Das resultierende AUI-Modell wird dann durch den *mimesis.ui.service* ⑤ in eine finale UI überführt. Abhängig von der angegebenen *Technologie* entsteht dabei eine plattformspezifische Seitenbeschreibung ⑥, die an die Antragstrecke geliefert wird.

Abbildungen 10.8 und 10.9 zeigen exemplarisch Ergebnisse der Generierung für Gesundheitsfragen. Die Navigation der Risikofragen ist in ein hierarchisches Menü des Vertreterpor-

tals integriert. Im Kundenportal erfolgt die Navigation sequentiell über *Weiter*-Schaltflächen innerhalb der Fragedialoge. Abbildung 10.8 zeigt die grundsätzlich unterschiedliche Darstellung der Plattformen. Abbildung 10.9 illustriert die semantikabhängige Darstellung von Interaktionselementen anhand der Fragen zur Konstitution. In Anhang A.6.1 befinden sich weitere Darstellungen zur Navigation sowie erzeugter Interaktionselemente.

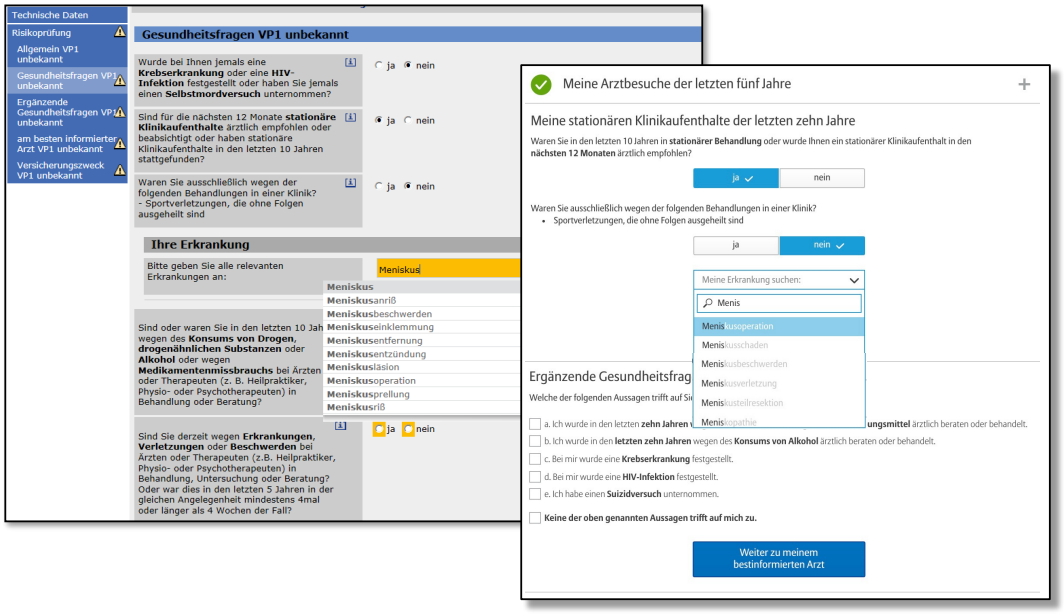


Abbildung 10.8. Ergänzende Gesundheitsfragen (Vertreter- und Kundenportal)

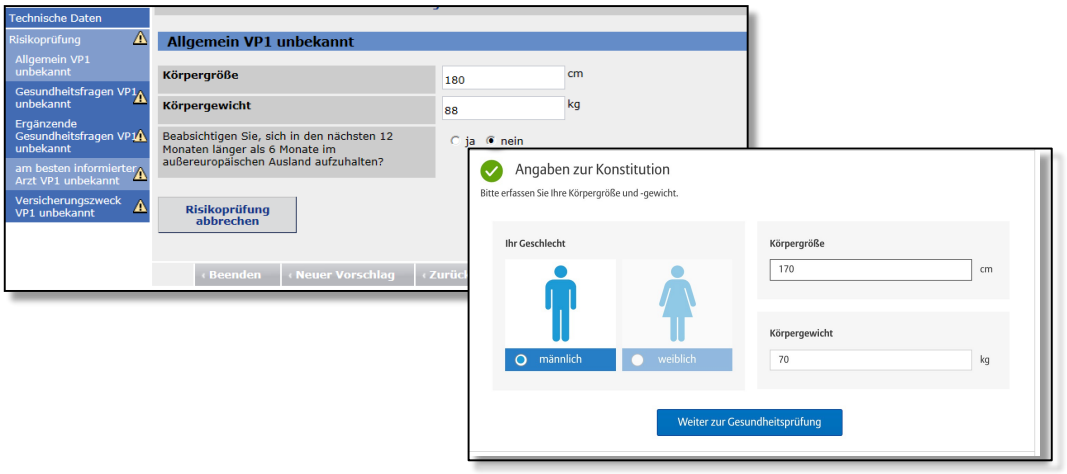


Abbildung 10.9. Fragen zur Konstitution (Vertreter- und Kundenportal)

Ergebnisse. Die Ergebnisse, die mit der dargestellten Architektur erzielt wurden, erfüllten die Anforderungen der umgesetzten Anwendungen vollständig, sodass das Verfahren in dieser Form heute produktiv zur Integration von Risikofragen in Antragstrecken eingesetzt wird. Das Verfahren zur Generierung erwies sich als flexibel zur Erzeugung von UIs, die in multiple Umgebungen/Infrastrukturen eingebettet werden und in ihren Interaktionselementen variieren. Dies reicht bis zur Erzeugung von spezialisierten Interaktionselementen, die auf unterschiedlichen Plattformen aufgrund der Semantik der zu erfassenden Daten zu verwenden sind (z.B. Verwendung von Metaphern und Ikonischen Darstellungen)

Zu Beginn der Evaluierung wurden lediglich UIs mit Standard-Interaktionselementen erzeugt, da der ARS unzureichende Informationen zur Semantik der Daten lieferte. Die in Abbildung 10.7 ④ dargestellte Transformation konnte jedoch dahingehend erweitert werden, dass die im mimesis Metamodell vorgesehenen Informationen (insbesondere die Angabe semantischer Annotationen) ergänzt werden konnten. Die Ergänzungen erfolgten über Heuristiken für spezifische Anwendungsfälle. Die erwünschten Ergebnisse konnten auf diese Weise für die umgesetzten Anwendungen erreicht werden. Eine generalisierte Lösung, z.B. durch automatische Semantik-Erkennung oder die Bereitstellung der Informationen durch den Backend-Dienst, kann in zukünftigen Arbeiten erfolgen.

10.4.2 Case Study: Freie Kombination von Dienstleistungen in einem *Dienstleistungs-Selektor*

Das Ziel dieser Evaluierung besteht im Nachweis der gemeinschaftlichen Nutzbarkeit frei verfügbarer Benutzungsschnittstellenbeschreibungen, deren Integrierbarkeit in einer gemeinsamen UI **(K.5)** sowie der Erzeugung von Instanzontologien zur Anbindung an Dienste eines Ökosystems **(K.6)**. Als Nachweismethode wird die Umsetzung einer Fallstudie (*Case Study*) gewählt, welche die Integration und Erzeugung von Instanz-Ontologien erfasster Daten an einem künstlichen Szenario zeigt.

Szenario: Dienstleistungs-Selektor

Als Szenario wurde eine *Dienstleistungs-Selektor*-Anwendung gewählt, welche dem Nutzer die Suche, Kombination und Beschreibung von Dienstleistungen aus beliebigen Domänen ermöglicht. Abbildung 10.10 zeigt die einzelnen Seiten der Anwendung.

Auf der Startseite der Anwendung Ⓐ sollen anhand eines Suchbegriffs ① Dienstleistungen gesucht und in einem *Warenkorb* zusammengestellt werden ②. Für die ausgewählten Module wird auf der Folgeseite Ⓑ eine gemeinsame Benutzungsschnittstelle zur Eingabe der Daten generiert ③. Den Abschluss der Eingabe bestätigt der Nutzer durch Betätigen einer Schaltfläche ④. Die erfassten Daten der einzelnen Module werden als Instanz-Ontologien auf einer Ergebnisseite Ⓒ dargestellt ⑤.

In Anhang A.6.2 befinden sich Verweise auf eine interaktive Version der Anwendung sowie ein Video, welches die Anwendung am Beispiel zur *Auswahl von Komponenten einer Reisebuchung* demonstriert.

Zielsetzung und Kriterien. Es ist nachzuweisen, dass der im Rahmen der Arbeit dargestellte Ansatz in diesem *generischen Szenario nutzbar* ist. Die Anwendung soll Komponenten beliebiger Domänen zur Kombination anbieten und eine kombinierte Benutzungsschnittstelle erzeugen, die in die Oberfläche integriert wird. Die Anwendungsmodelle der Komponenten sollen dabei dezentral zur Verfügung gestellt werden. Die in der Oberfläche erfassten Daten sollen anschließend als Instanz-Ontologien aufbereitet werden.

Entsprechend ergeben sich die Kriterien, mit denen das Ergebnis bewertet werden kann. Mit der Evaluierung müssen (1) die Auswahl beliebiger Anwendungsbeschreibungen aus beliebigen Quellen, (2) die Zusammenfassung beliebiger Anwendungsmodelle in einer Benutzungsschnittstelle und (3) die Repräsentierbarkeit der erfassten Daten als Instanz-Ontologien

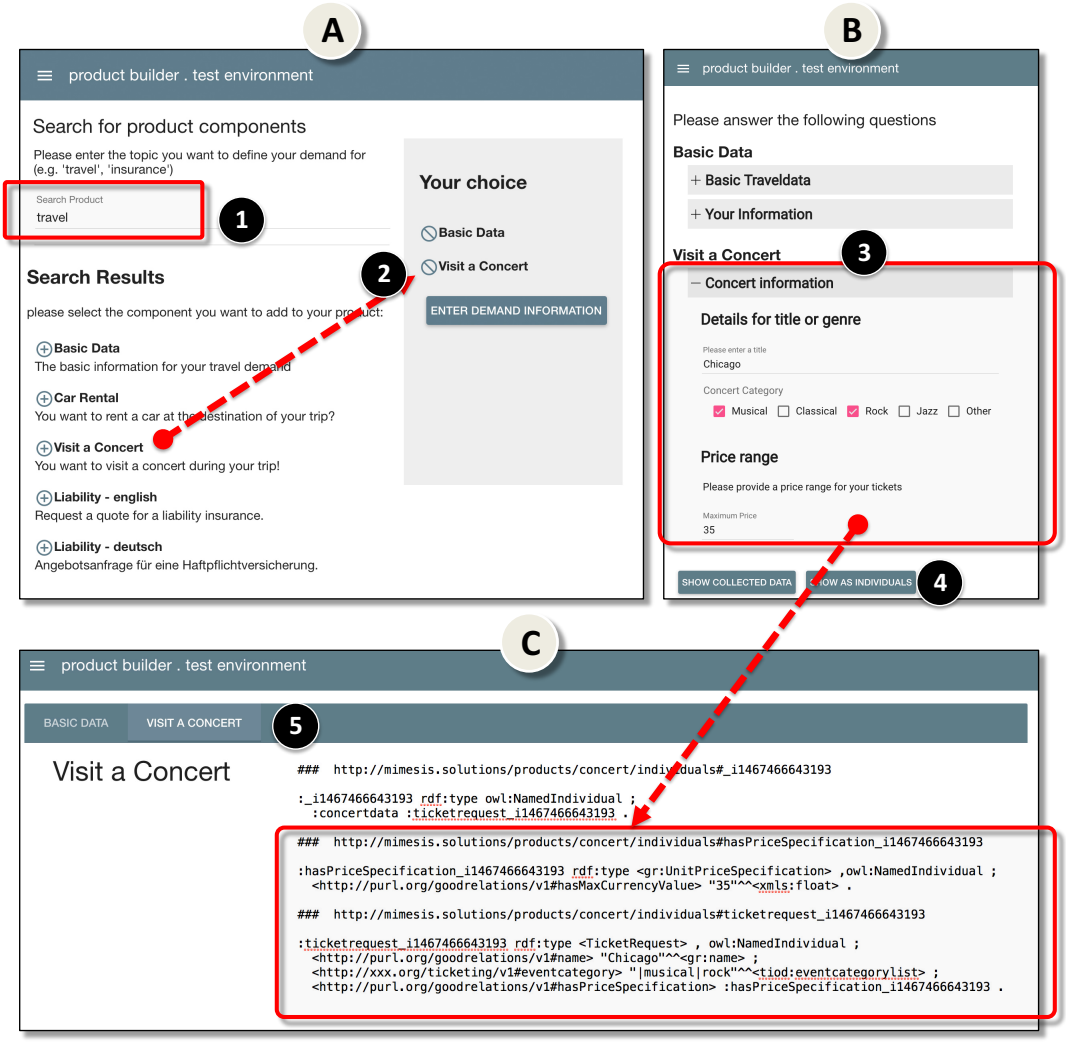


Abbildung 10.10. Benutzungsschnittstelle des *Service Selektor*-Demonstrators

nachgewiesen werden. Zur Umsetzung soll die Anwendung (4) neben der URI des Anwendungsmodells und ggf. Varianten keine weiteren Kenntnisse erfordern.

Durchführung. Die technische Architektur für den Versuchsaufbau ist in Abbildung 10.11 dargestellt. Die Umsetzung folgt der in Abschnitt 10.2.2 dargestellten Anwendungsarchitektur und ist in einen applikationsspezifischen Anteil und das Benutzungsschnittstellen-Ökosystem (*UI-Ökosystem*) als gemeinschaftlich genutzten (*shared*) Bereich aufgeteilt.

Als Themenbereich für den Demonstrator wurde die *Buchung einer Reise* umgesetzt. Hierfür wurden für *Reise-bezogene Produkte* Anwendungsmodelle erstellt, die mit Korrelationsinformationen für Dienst-Ontologien angereichert wurden ①. Zur Umsetzung der ersten Seite der Anwendung (Suche) besitzt der Demonstrator eine eigene Datenhaltung für Dienstleistungsbeschreibungen, die bei der Eingabe eines Suchbegriffs durchsucht wird. Die Beschreibungen enthalten die URIs der zu verwendenden Anwendungs- und Variantenmodelle. Der Nutzer stellt im Warenkorb eine Liste gewünschter Komponenten zusammen ②. Die Generierung der Benutzungsschnittstelle erfolgt anschließend anhand der URIs der Komponenten ③. Hierzu wird der *mimesis.ui.service* gerufen, der aus der Liste übergebener URIs eine FUI

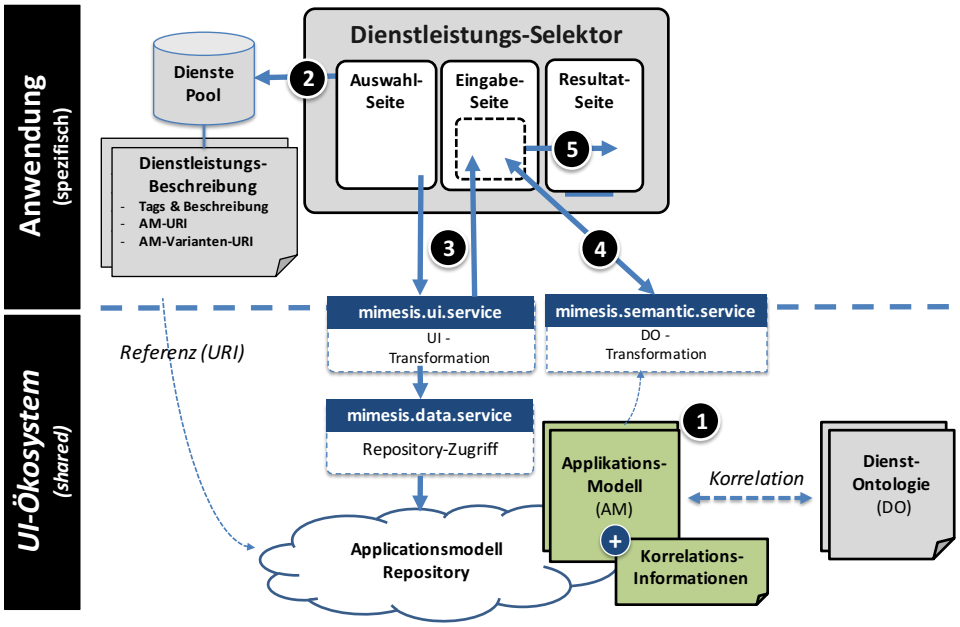


Abbildung 10.11. Architektur des *Service Selektors*

erzeugt, die in die zweite Seite der Anwendung integriert wird. Nach Abschluss der Eingabe werden für jede Komponente die URI und die Instanzdaten an den *mimesis.semantic.service* zur Instanzdatentransformation übergeben ④. Diese erstellt eine Instanz-Ontologie, die abschließend in der Ergebnisseite dargestellt wird ⑤.

In Anhang A.6.2 befinden sich weitere Informationen und Verweise zu lauffähigen Artefakten des Demonstrators, anhand derer die Ergebnisse nachvollzogen werden können.

Ergebnisse. Obwohl der Demonstrator lediglich ein *Proof-of-concept* für ein künstliches Szenario darstellt, konnte damit die Erfüllung wesentlicher Anforderungen an den Ansatz nachgewiesen werden. Es konnte insbesondere demonstriert werden, dass (1) Komponenten aus **unterschiedlichen Quellen** eines UI-Ökosystems gemeinschaftlich nutzbar sind und (2) dynamisch beliebige **Komponenten unterschiedlicher Domänen** in einer Benutzungsschnittstelle zusammengeführt werden können. Es konnte weiterhin gezeigt werden, dass (3) aus den Nutzereingaben ein Ergebnis erzeugt werden kann, das **Dienste eines Dienst-Ökosystems** verarbeiten können. Der Demonstrator zeigt zudem, dass (4) in der nutzenden Anwendung kein Detailwissen zu den Komponenten oder Domänen erforderlich ist. Die Komponenten können dezentral zur Verfügung gestellt und daraus lauffähige Anwendungen erstellt werden. Diese liefern ein Resultat, das von angeschlossenen Diensten verwertbar ist.

10.4.3 Action Research: Shareable UIs für komplexe Produktanfragen in Distributed Marketspaces

Das Ziel dieser Evaluierung besteht im Nachweis der gemeinschaftlichen Nutzbarkeit frei verfügbarer Benutzungsschnittstellenbeschreibungen, deren Integrierbarkeit in einer gemeinsamen UI **(K.5)** sowie der Erzeugung von Instanzontologien zur Anbindung an Dienste eines Ökosystems **(K.6)** in einem bestehenden Umfeld. Die Evaluierung erfolgte in Zu-

sammenarbeit im Rahmen einer Dissertation zum Thema verteilte Marktplätze (*Distributed Market Spaces*), die an der DHBW-Mannheim von Mirjana Radonjic-Simic durchgeführt wird [168, 183–185].

Kontext: *Distributed Marketspaces*

Distributed Market Spaces (DMS) beschreiben einen Ansatz, *Produkte* (z.B. Dienstleistungen, Güter) im Sinne einer *Peer-to-Peer*-Beziehung zwischen Anbietern (*Provider*) und Konsumenten (*Consumer*) zu vermitteln. Radonjic-Simic erweitert dieses Konzept auf komplexe Produktanfragen (*Complex Products*) und stellt eine Gesamtarchitektur und Prozesse dar, welche die Abwicklung solcher Anfragen in einem verteilten Umfeld über mehrere Marktplätze hinweg ermöglicht.

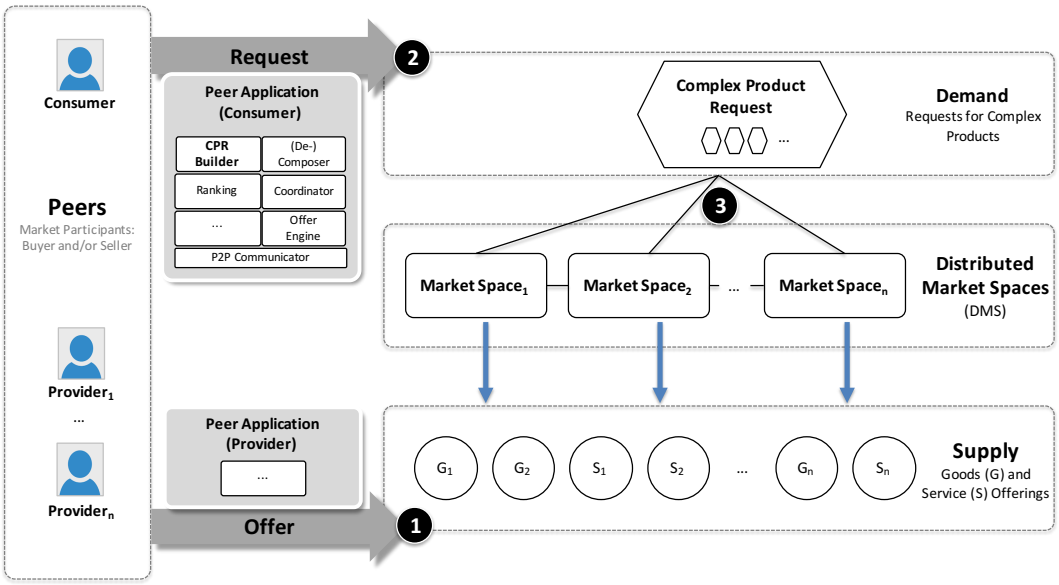


Abbildung 10.12. Überblick *Distributed Marketspace*

Abbildung 10.12 (mit Änderungen aus [101]) stellt den Ansatz fokussiert auf die Aufgabenstellung vereinfacht dar: *Provider* beschreiben Angebote für einzelne *Produkte* und veröffentlichen sie auf einem Marktplatz ①. *Consumer* stellen Anfragen für Produkte, die sich aus mehreren Teilprodukten zusammensetzen können (*Complex Product Requests*, CPR) ②. Zur Erstellung eines Angebots für das komplexe Produkt wird ein vom *Consumer* erstellter CPR in einzelne, auf seinen Nutzerkontext (z.B. Vorlieben, Einschränkungen, Gewohnheiten) optimierte Anfragen zerlegt ③. Die einzelnen Anfragen werden mit Angeboten eines oder mehrerer Marktplätze abgeglichen. Aus den Ergebnissen wird dann ein zu den Anfragen und dem Nutzerkontext passendes Gesamtangebot ermittelt, das in weiteren Schritten bis hin zur Abwicklung des Kaufs vom Marktplatz verarbeitet wird. Sowohl Angebote als auch Anfragen werden in diesem Szenario mit semantischen Technologien beschrieben. Angebote werden über Instanzen einer Angebots-Ontologie (*Supply Onotlogy*) für bestimmte Produktgruppen und Anfragen über entsprechende Anfrage-Ontologien (*Demand Ontology*) spezifiziert (vgl. [101]). Der Marktplatz ist dadurch in der Lage, Angebote und Anfragen in Beziehung zu bringen.

Zur Integration der beteiligten Rollen (*Peers*) in diesem Prozess werden Benutzungsschnittstellen benötigt, die einerseits *Providern* das Erstellen von Angeboten, andererseits *Con-*

sumern die Eingabe der komplexen Produktanfragen ermöglichen. Hierfür sind rollenabhängige Anwendungen (*Peer-Applications*) vorgesehen. Abbildung 10.12 zeigt die Komponenten einer *Peer Application* für die *Consumer*-Rolle. Sie enthält eine *Complex Product Request Builder*-Komponente (CPRB), welche die Eingabe der Daten anzufragender Produkte ermöglicht und als Resultat einen CPR erzeugt. Der CPR wird in Teilanfragen zerlegt (*De-/Composer*), Angebote von teilnehmenden Marktplätzen eingeholt (*Coordinator*, *Offer Engine*, *P2P-Communicator*) und ein Gesamtangebot erstellt (*OfferEngine*, *Ranking*). Zu weiterführenden Details des Ansatzes wird auf die Arbeiten von Radonjic-Simic et al. [168, 183, 184] und [101] verwiesen.

Zielsetzung und Kriterien. Das Ziel der Evaluierung besteht darin, eine generische Lösung für die CPRB-Komponente zu erstellen. Die *Peer Application* soll dabei die Auswahl von Produkten bereitstellen, für welche Eingabedialoge zur Erfassung der Anfrageinformationen automatisch erzeugt und dargestellt werden. Die Anwendungsmodelle der Komponenten sollen dabei dezentral zur Verfügung gestellt werden. Nach Abschluss der Eingabe sollen die erfassten Informationen als Anfrage-Ontologie-Instanzen bereitgestellt werden. Die *Peer Application* verwendet zur Erstellung eines Angebots diese Daten zur Kommunikation mit den Marktplätzen des DMS.

Die Kriterien zur Erfüllung sind, dass (1) die Benutzungsschnittstelle aus *SharedUIs* aus beliebigen Quellen aufgebaut wird, die von der *Peer Application* über ihre URI vorgegeben werden, (2) die erzeugte Benutzungsschnittstelle in die *Peer Application* integriert wird und (3) die Eingaben in Form einer Instanz für konkrete Anfrage-Ontologien bereitgestellt werden.

Die **Verbesserung**, die durch Anwendung der Artefakte erreicht wird, besteht in der generischen Erstellung der Benutzungsschnittstellen für beliebige Kombinationen vom DMS bereitgestellter Produkte. Sofern für Anfrage-Ontologien Anwendungsbeschreibungen aus einer Quelle bezogen werden können, sind automatisiert Eingabedialoge erstellbar, die CPRs erzeugen können. Mit diesen Mitteln können Marktplätze beliebige Produktkategorien verarbeiten, ohne dass Detailwissen zu den einzelnen Produkten erforderlich ist.

Durchführung. Zur Illustration des Zielbildes zeigt Abbildung 10.13 die in die *Peer Application* integrierte Benutzungsschnittstelle zur Eingabe der Anfragedaten für ein komplexes Produkt. Sie zeigt den Zustand, nachdem die Produkte ausgewählt, die UI angefordert und in die Anwendung integriert wurde. Unter den in Anhang A.6.3 referenzierten Verweisen befindet sich ein Video, welches den Anwendungsfall einer Reisebuchung von der Auswahl der Produkte bis hin zur Darstellung des resultierenden Angebots demonstriert.

Die technische Architektur für den Versuchsaufbau ist in Abbildung 10.14 dargestellt. Die Umsetzung folgt der in Abschnitt 10.2.2 dargestellten Anwendungsarchitektur. Sie zeigt die Nutzung der Artefakte und Komponenten für das DMS-Szenario. Hier wird der *mimesis.ui.service* zur Erzeugung der finalen UIs, der *mimesis.data.service* zum Zugriff auf Anwendungsmodelle sowie der *mimesis.semantic.service* zur Erzeugung von Instanz-Ontologien für Eingabedaten verwendet.

Zur Durchführung der Evaluation wurden Anwendungsmodelle zur Erfassung von Anfragedaten für unterstützte Produkte des DMS erstellt. Diese basierten auf den Daten, die durch entsprechende Anfrage-Ontologien vorgegeben sind und wurden mit Korrelationsinformationen zur Herleitung einer Anfrage-Ontologie-Instanz (AOI) ergänzt. Im Folgenden wurde

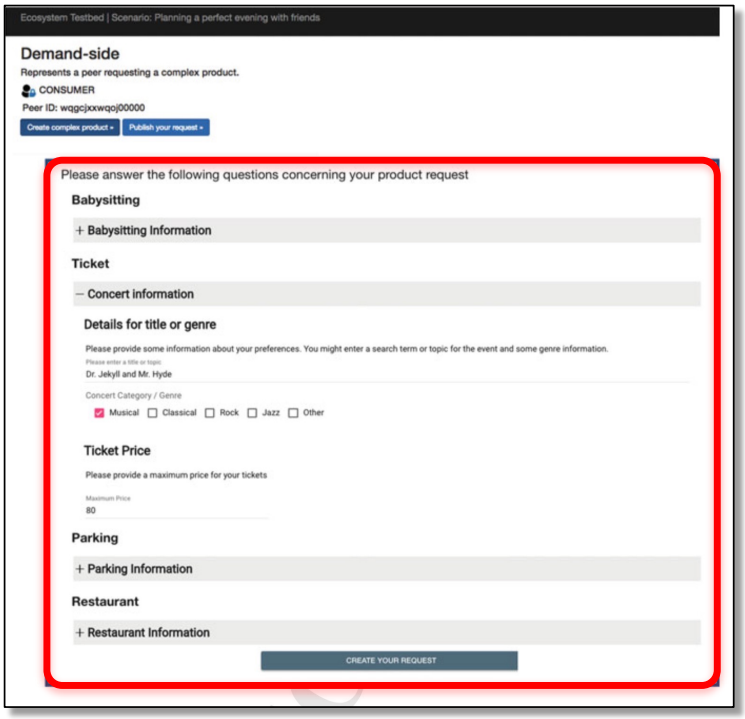


Abbildung 10.13. Eingabe der Anfragedaten in der *Peer-Application*

die CPBR-Komponente als Teil der *Peer Application* umgesetzt. Die Aufgabe der Komponente liegt in der Erfassung der Daten zu einem komplexen Produkt und der Erstellung einer komplexen Produktanfrage in Form einer Anfrage-Ontologie-Instanz.

In Abbildung 10.14 ist das Zusammenspiel dargestellt. Im Rahmen der *Peer Application* erfolgt die Auswahl der Produkte, aus denen das komplexe Produkt zusammengesetzt sein soll. Für jedes Teilprodukt wird bestimmt, welches Anwendungsmodell zu verwenden ist (Angabe der Modell-URI) ①. Die URIs werden der CPRB-Komponente übergeben, die daraus mittels des *mimesis.ui.service* eine kombinierte Benutzungsschnittstelle erzeugt ②. Diese wird in die Oberfläche der *Peer Application* integriert. Nach Abschluss der Eingabe erzeugt die CPRB-Komponente aus den Eingabedaten mittels des *mimesis.semantic.service* für die Teilprodukte Anfrage-Ontologie-Instanzen ③. Diese werden von der *(De)-Composer*-Komponente der *Peer Application* für Anfragen an den DMS verwendet ④. Die Ergebnisse der Anfragen werden anschließend von der *Peer Application* dargestellt bzw. weiterverarbeitet.

In Anhang A.6.3 werden Verweise auf die erstellten Anwendungsbeschreibungen mit den entsprechenden Korrelationsinformationen angegeben.

Ergebnisse. Mit dem Einsatz der Artefakte der Arbeit im Umfeld von *Distributed Market Spaces* konnt die Erfüllung grundlegender Anforderungen des Ansatzes für *SharedUIs* in einem realen Szenario nachgewiesen werden. Die Evaluation demonstriert, dass Anwendungsmodelle unterschiedlicher Domänen aus multiplen Quellen gemeinschaftlich nutz- und kombinierbar sind (1),(2). Sie zeigt zudem, dass das angegebene Verfahren zur Herleitung von Ontologie-Instanzen zur Anbindung von Diensten in einem realitätsnahen Umfeld tragfähig ist und damit Dienste in einem Dienst-Ökosystem generisch angebunden werden können (3).

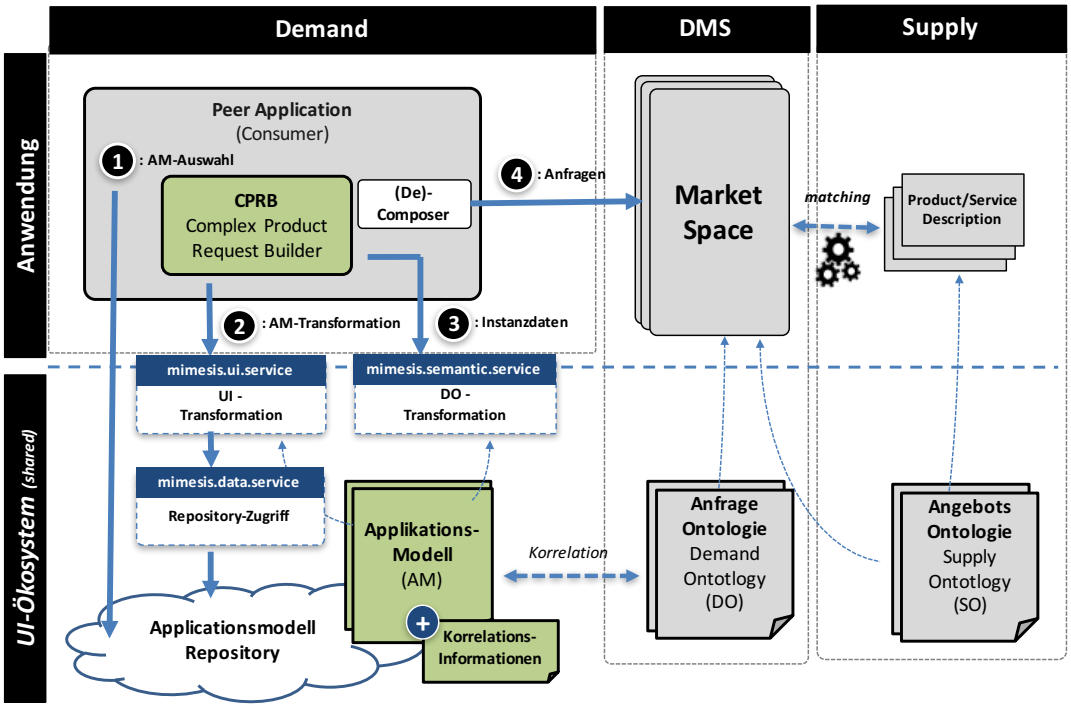


Abbildung 10.14. Lösungsarchitektur - DMS und CPRB

Durch Verwendung einer *Action Research*-Evaluation konnte die Nutzbarkeit des Ansatzes für *SharedUIs* nachgewiesen werden. Die Artefakte konnten auf das reale Szenario angewendet werden. Dabei wurde demonstriert, dass durch den *SharedUI*-Ansatz Dienste eines Dienst-Ökosystems integriert werden können, deren fachliche Inhalte der nutzenden Anwendung nicht bekannt sein müssen. Dies stellt eine neue Form der Anwendungsentwicklung dar, die nach meinem besten Wissen mit keinem bestehenden Ansatz durchgängig umgesetzt werden kann.

10.5 Ergebnisse und Bewertung

Die im Rahmen des Kapitels durchgeführten Untersuchungen demonstrieren, dass der Ansatz der Arbeit zur Lösung der Problemstellung beiträgt und praktisch eingesetzt werden kann. Insbesondere konnten die in Abschnitt 10.1 aufgestellten Bewertungskriterien **(K.1)-(K.6)** erfüllt werden.

Funktionale Bewertung. In Abschnitt 10.2 wurde durch prototypische Implementierung der Lösungsartefakte der Arbeit die **Machbarkeit** des Ansatzes nachgewiesen. Die vorgeschlagenen Artefakte des Ansatzes konnten in technische Komponenten umgesetzt werden. In Abschnitt 10.3 wurde gezeigt, dass die entwickelten Artefakte auf die adressierten Problemstellungen der Arbeit angewendet werden können und diese lösen (**Anwendbarkeit**). Es wurde nachgewiesen, dass mittels der Artefakte lauffähige Benutzungsschnittstellen in Varianten automatisch erzeugt und in neuen Anwendungen wiederverwendet werden können. Es erfolgte anhand realitätsnaher Szenarien der Nachweis, dass

- zu manuell erstellten funktional gleichwertige Benutzungsschnittstellen **(K.1)**

- inhaltliche Varianten von Benutzungsschnittstellen **(K.2)**
- Kompositionen bestehender Benutzungsschnittstellen **(K.3)**
- technische Varianten vollständig automatisiert **(K.4)**

erzeugt werden können. Zudem erfolgte der Nachweis, dass die Artefakte in einer universellen Repräsentation beschrieben werden können, die eine gemeinschaftliche Nutzung ermöglicht. In Abschnitt 10.4 wurde nachgewiesen, dass der Ansatz auf konkrete Anwendungsfälle angewendet werden kann (**Nutzbarkeit**) und hier zu Verbesserungen bei der Erstellung der betrachteten Szenarien führt. Es wurde gezeigt, dass die Kriterien (K.1)-(K.4) auch in einem realen Kontext erfüllt werden. Zusätzlich wurde die Erfüllung der Kriterien (K.5) und (K.6) demonstriert und gezeigt, dass

- gemeinschaftlich nutzbare Benutzungsschnittstellen beschrieben werden können, die als Komponenten in Anwendungen Dritter integrierbar sind **(K.5)**
- Benutzungsschnittstellen erzeugt werden können, die an beliebige Dienste eines Dienst-Ökosystems angebunden werden können **(K.6)**

Die hierzu betrachteten Anwendungsfälle entstammen nicht ausschließlich dem akademischen Bereich, sondern im Sinne eines *Action Research*-Vorgehens auch dem realen Umfeld der Allianz Deutschland AG. Damit konnte die Nutzbarkeit des Ansatzes am *lebenden Beispiel* demonstriert werden.

Qualitative Bewertung der Ergebnisse. Wie bereits in Abschnitt 4.2 dargestellt, erfolgten *ex-post* Untersuchungen zur Qualität der erzeugten Benutzungsschnittstellen im Rahmen der Arbeit in unstrukturierten Experten-Gesprächen. Obwohl die Bewertung nicht systematisch durchgeführt wurde, kann sie dennoch als Indikator für die Qualität der Ergebnisse dienen.

Die Experten waren bei der Auswahl der zur Analyse herangezogener Anwendungen beteiligt und trugen zur Bereitstellung eines möglichst breiten Spektrums an Interaktionsmustern bei (vgl. Abschnitt 5.1, Datengrundlage). Die in der Validierung umgesetzten Benutzungsschnittstellen folgten dieser Auswahl, wodurch eine weitgehende Abdeckung der verbreiteten Muster angenommen werden kann. Im Zuge der Validierung wurde nachgewiesen, dass die Lösungen den vorgefundenen Mustern entsprechen, was auf deren qualitative Gleichwertigkeit hindeutet.

Insbesondere bei der Evaluation der Anwendungen der Risikoanalyse als *Action Research* stand die Qualität der erzeugten Benutzungsschnittstellen im Fokus, da die Resultate in den produktiven Betrieb überführt wurden. An der Bewertung waren unterschiedliche Personengruppen beteiligt (z.B. Entwickler, Architekten, Projektleiter, Stakeholder), die unterschiedliche Expertisen beisteuerten. Neben der Bewertung der Benutzungsschnittstellen anhand der Kriterien der Arbeit konnte hier mit bereits existierenden Anwendungen im Vermittlerumfeld verglichen werden. Die Lösung entsprach dabei den gesteckten Anforderungen und führte zusätzlich zu qualitativen Verbesserungen der Benutzungsschnittstellen: so konnten für unterschiedliche Nutzergruppen spezifische Eingabeelemente generiert werden, welche die Anwendung benutzerfreundlicher gestalteten - und damit deren Qualität verbesserten.

Die Experten-Bewertungen und der Einsatz in der *realen Welt* im Besonderen sind ein deutlicher Indikator dafür, dass qualitativ hochwertige Benutzungsschnittstellen mit dem Ansatz erzeugt werden können.

Verbesserungen durch die Lösung. Die Validierung und Evaluation zeigte die Verbesserungen auf, die sich für die Entwicklung und die Qualität ergeben. Alle Benutzungsschnittstellen konnten vollständig automatisiert generiert werden und Varianten sowie beliebige Kombinationen bestehender Anwendungen waren mit geringem Aufwand erzeugbar.

In den Szenarien der Validierung war die Verbesserung offensichtlich, da ein einziges Modell für alle Varianten genutzt und in Kombinationen wiederverwendet werden konnte. Da die Validierungen mit auf die Lösung zugeschnittenen Szenarien durchgeführt wurden, ist dies zu erwarten.

Die Verbesserungen ergaben sich jedoch auch bei den konkreten Anwendungsfällen der Evaluation. So wäre das Szenario der verteilten Marktplätze ohne eine freie Kombinierbarkeit und Integration in verschiedene Technologien nur schwer möglich. Für das Szenario der Risikoanalyse-Dialoge konnte die Verbesserung direkt in der Praxis nachgewiesen werden. Hier werden die Risikofragen bereits im Kunden- und Vermittlerumfeld wiederverwendet und in die entsprechenden Benutzungsschnittstellen integriert. Die automatische Generierung beliebiger Frage-Dialoge in die entsprechenden Technologien führte dazu, dass in der Folge UIs für neu hinzugekommene Risiko-Abläufe automatisch auf allen Plattformen verfügbar waren - dies war bis dahin unmöglich. Bisher war die Bereitstellung von UIs eine manuelle Aufgabe, welche die Neuerstellung einer Anwendung zur Folge hatte.

Ergebnis

Die Implementierung, Validierung und Evaluation konnten zeigen, dass die Ziele der Arbeit erreicht wurden, dass die vorgestellte Lösung auf realitätsnahe und reale Probleme anwendbar ist und dort zu konkreten Verbesserungen führt. Durch den Einsatz im Umfeld der Allianz Deutschland AG konnte zudem die Anwendbarkeit des Ansatzes auf konkrete Probleme in der Praxis demonstriert und dessen Nutzung im produktiven Umfeld nachgewiesen werden. Hier führt die Lösung bereits heute zu signifikanten Verbesserungen und Optimierung in der Anwendungsentwicklung.

11. Zusammenfassung, Bewertung, Ausblick und Fazit

Die Arbeit präsentiert einen Ansatz zur Modellierung und automatischen Generierung dialogbasierter Benutzungsschnittstellen für digitale Dienste. Er schafft die Möglichkeit, Menschen mit unterschiedlichen Anforderungen und Bedürfnissen an der Digitalisierung teilhaben zu lassen - mit minimalen Aufwänden für die Anbieter digitaler Dienstleistungen.

Ein Ziel der Digitalisierung ist es, IT-gestützte Dienstleistungen aus verschiedenen fachlichen Domänen zu kombinieren und unterschiedlichen Nutzergruppen zugänglich zu machen. Die automatische Generierung von Benutzungsschnittstellen kann hier einen wesentlichen Beitrag leisten, bietet bisher jedoch lediglich Teillösungen (vgl. Abschnitt 1.2). Insbesondere besteht ein Mangel an Ansätzen zur automatisierten Erstellung nutzungsspezifischer Varianten und deren Kombination zu funktional höherwertigen Anwendungen. Die Wiederverwendung existierender Benutzungsschnittstellen ist meist nicht möglich, da sie mit spezifischen Technologien umgesetzt sind und nicht im Sinne einer nahtlosen Integration an ein neues Umfeld anpassbar sind. Zudem existieren keine Konzepte zur Bereitstellung gemeinschaftlich nutzbarer Benutzungsschnittstellen.

Im Verlauf der Arbeit wird ein modellgetriebenes Vorgehen zur Lösung dieser Herausforderungen entwickelt, mit dem folgende Ziele erreicht werden (vgl. Abschnitt 1.3):

1. Vereinfachung der Erstellung und Wartung von Benutzungsschnittstellen
2. Geringere Komplexität und Redundanz bei der Modellierung von Varianten
3. Automatische Erzeugung nicht-trivialer Benutzungsschnittstellenvarianten
4. Erhöhte Wiederverwendbarkeit der Modelle durch Kombinierbarkeit in variierenden Szenarien
5. Gemeinschaftliche Nutzbarkeit der Modelle in Dienst-Ökosystemen

Im Folgenden werden die Ergebnisse entlang der Forschungsfragen der Arbeit zusammengefasst und anhand der Ziele der Arbeit, der Neuerungen und Wirkungen bewertet. Abschließend wird ein Ausblick auf offene Fragestellungen sowie weiterführende Themen und Anwendungsbereiche der Arbeit gegeben.

11.1 Zusammenfassung der Ergebnisse und Beiträge

Die vorgeschlagene Lösung besteht darin, ein technologieneutrales, *datenzentriertes Modell* als Grundlage zur Modellierung von Benutzungsschnittstellen zu verwenden. Es beschreibt die zu erfassenden Daten und enthält alle zur automatischen Generierung finaler UIs erforderlichen Informationen. Derartige Modelle können zu komplexeren Anwendungen kombiniert und optional um Variantenbeschreibungen ergänzt werden, welche redundanzfrei die Unterschiede zwischen Varianten beschreiben. Da die Modellartefakte hinsichtlich der Generierung vollständig beschrieben sind, können sie darüber hinaus in einer verteilten Umgebung zur Wiederverwendung bereitgestellt und gemeinschaftlich genutzt werden.

Die Entwicklung der Lösung erfolgt dabei entlang der in Abschnitt 1.3 gestellten Forschungsfragen:

(FF.1): Wie können dialogbasierte Benutzungsschnittstellen und Varianten technologieneutral beschrieben werden?

Es wird in Form einer Analyse gezeigt, dass Benutzungsschnittstellen und deren Varianten technologieneutral über ein datenzentriertes Modell beschreibbar sind und die hierzu notwendigen Informationen identifiziert (Kapitel 5). Das Ergebnis besteht in einer systematischen Darstellung der zur Herleitung dialogbasierter Anwendungen benötigten Informationen.

Das Ergebnis leistet einen Beitrag zur Spezifikation der Anforderungen an die Modellierung dialogbasierter Anwendungen. Die Neuerung besteht in der Fokussierung auf die erfassten Daten einer Anwendung und der Technologieneutralität der Informationen, aus denen Benutzungsschnittstellen für multiple Technologien ableitbar sind.

(FF.2): Wie können Benutzungsschnittstellen und Varianten modelliert werden?

Auf den Ergebnissen aufbauend wird ein Metamodell zur Beschreibung dialogbasierter Benutzungsschnittstellen präsentiert sowie eine Methodik zur Spezifikation inhaltlicher Varianten dargestellt (Kapitel 6). Die Lösung reduziert die Modellierung auf zwei Artefakte, die sich zur automatisierten Erzeugung inhaltlicher und technischer Varianten eignen.

Die Beiträge liegen in einem Metamodell zur datenzentrierten Modellierung von Benutzungsschnittstellen sowie einer Methodik und einem Metamodell zur Spezifikation inhaltlicher Varianten. Die Neuerung liegt dabei in der Modellierung vollständig generierbarer Benutzungsschnittstellen über ein zentrales Artefakt (*Anwendungsmodell*), auf dem redundanzfrei Varianten spezifiziert werden (*Variantenmodell*).

(FF.3): Wie können Anwendungsmodelle zu komplexeren Anwendungen kombiniert werden?

Auf dem Metamodell aufbauend wird die Kombination von Anwendungsmodellen zu komplexeren Anwendungen sowie eine Methode zu deren Anpassung an eine neue Nutzung dargestellt (Kapitel 7). Es wird gezeigt, dass der datenzentrierte Ansatz zur Modellierung anpassbarer Kompositionen geeignet ist und damit bestehende Modelle zu höherwertigen Anwendungen kombinierbar sind.

Das Ergebnis leistet einen Beitrag zur Modellierung wiederverwendbarer Benutzungsschnittstellen. Die Neuerung liegt insbesondere in der nahtlosen Integration bestehender Komponenten in ein neues Umfeld und einer Methodik zu deren Anpassung an variierende Nutzungsszenarien zum Zeitpunkt der Wiederverwendung.

(FF.4): Wie lassen sich aus einem Anwendungsmodell technische Varianten automatisch herleiten?

Es wird ein Verfahren zur Generierung der Kompositionen und Varianten entwickelt (Kapitel 8). Dabei wird gezeigt, dass aus den Modellen vollständig automatisiert lauffähige Anwendungen für verschiedene Interaktions- und Nutzungskontexte erzeugbar sind. Das Verfahren basiert auf den Prinzipien des CAMELEON Referenz-Frameworks und konkretisiert diese für die Generierung von UIs aus datenzentrierten Modellen.

Der Beitrag besteht in der Erweiterung der bestehenden Lösung um Schritte zur automatisierten Komposition und Erzeugung von Varianten. Die Neuerung liegt dabei in der vollständig automatisierten Generierung ohne manuelle Schritte.

(FF.5): Wie können Benutzungsschnittstellen in Dienst-Ökosystemen gemeinschaftlich wiederverwendet werden?

Die Ergebnisse der vorangegangenen Schritte werden auf einen Ansatz zur Wiederverwendung von Benutzungsschnittstellen angewendet (Kapitel 9). Hierzu wird ein Konzept zur gemeinschaftlichen Nutzung von *Shared UIs* in einem Dienst-Ökosystem vorgeschlagen. Es wird eine Architektur vorgestellt, welche die oben genannten Modelle und Verfahren als wiederverwendbare Komponenten in einer dezentralen Infrastruktur zur Verfügung stellt. Die Komponenten (statische und funktionale) können dabei von unterschiedlichen Anbietern bereitgestellt werden.

Für die Modelle wird hierzu eine universelle und verteilbare Repräsentation in Form von Ontologien vorgestellt, die über dezentrale Dienste in finale Benutzungsschnittstellen transformiert werden. Zudem wird eine Lösung zur Überführung erfasster Daten in Eingabedaten für standardisierte Ökosystem-Dienste dargestellt.

Der Beitrag besteht in einer umfassenden Lösung zur verteilten Bereitstellung und gemeinschaftlichen Nutzung wiederverwendbarer Benutzungsschnittstellen. Die Neuerung liegt in der Nutzung einer minimalen Zahl an Artefakten zur Beschreibung als Grundlage zur vollständig automatisierten Generierung funktionsfähiger Komponenten. Diese sind kombinierbar und in multiplen fachlichen und technischen Kontexten nutzbar.

Der Ansatz unterscheidet sich von bestehenden Ansätzen insbesondere durch die Verwendung des datenzentrierten Modells und dessen universeller Repräsentation als Ontologien. Dies ermöglicht die gemeinschaftliche Nutzung und Integration in multiple Umgebungen. Die Beschreibungen werden zu frei verfügbarem Wissen (*shared knowledge*) und sind damit für jeden Teilnehmer eines Ökosystems zugänglich und interpretierbar.

Praktikabilität: Sind die gefundenen Lösungen tragfähig?

Es wird gezeigt, dass die entwickelten Artefakte, Verfahren und die Architektur einerseits technisch umsetzbar, andererseits auf reale bzw. realitätsnahe Problemstellungen anwendbar sind (Kapitel 10). Zum Nachweis der Umsetzbarkeit werden die Artefakte des präsentierten Ansatzes prototypisch implementiert und die Resultate hinsichtlich der in der Arbeit gestellten Anforderungen validiert. Die Nutzbarkeit des Ansatzes wird über Fallstudien und den Einsatz in einem realen Umfeld i.S. des *Action Research*-Prinzips demonstriert.

11.2 Bewertung der Ergebnisse

Die Bewertung der Ergebnisse erfolgt in den folgenden Abschnitten unter drei Aspekten:

- Erreichung der Ziele der Arbeit (Abschnitt 11.2.1)
- Neuartigkeit der Lösung (Abschnitt 11.2.2)
- Auswirkungen auf die Entwicklung und Endnutzer (Abschnitt 11.2.3)

Es wird einerseits dargestellt, inwiefern die selbst gesteckten Ziele der Arbeit durch die Lösung erreicht wurden und andererseits diskutiert, welche Neuerungen dem Stand der Forschung hinzugefügt werden und wie sich diese auf das Entwicklung und Nutzer auswirken.

11.2.1 Erreichung der Ziele der Arbeit

Mit dem vorgestellten Ansatz werden die in Abschnitt 1.3 formulierten Ziele der Arbeit erreicht. Für die einzelnen Ziele wird dies durch folgende Lösungseigenschaften umgesetzt:

Vereinfachung der Erstellung und Wartung von Benutzungsschnittstellen

Im Vergleich zu bestehenden Ansätzen vereinfacht die vorgestellte datenzentrierte Modellierung die Erstellung und Wartung von Anwendungen, da sie nur ein einziges, technologieneutrales Artefakt zur Beschreibung erfordert. Dieses enthält alle zur Generierung notwendigen Informationen und ist aufgrund der Datenzentrierung dennoch vergleichsweise einfach zu erstellen.

Geringere Komplexität und Redundanz bei der Modellierung von Varianten

Die Komplexität zur Erstellung und Wartung von Varianten wird durch die vorgestellte differentielle Modellierung über nur ein weiteres Artefakt ebenfalls signifikant reduziert, da lediglich Unterschiede zum Ausgangsmodell beschrieben werden. Modifikationen des Modells, die z.B. bei der Wartung erfolgen, wirken sich automatisch auch auf Varianten aus. Dies reduziert Anpassungsaufwände auf ein notwendiges Minimum (z.B. Anpassen von Modifikationen für entfallene oder hinzugefügte Elemente sowie Umstrukturierungen).

Automatische Erzeugung nicht-trivialer Benutzungsschnittstellenvarianten

Das präsentierte Generierungsverfahren eignet sich zur vollständig automatischen Erzeugung finaler Benutzungsschnittstellen. Hierzu werden lediglich das Anwendungs- und ein optionales Variantenmodell benötigt, um daraus ohne manuelle Schritte eine Anwendung für unterschiedliche Ablaufumgebungen zu erzeugen. Da das Anwendungsmodell Informationen zur Semantik der darzustellenden Daten beinhaltet, können spezifische Interaktionselemente und damit eine nicht-triviale Benutzungsschnittstelle erzeugt werden.

Dies trägt zusätzlich zur Vereinfachung der Modellierung von Anwendungen bei. Da das Generierungsverfahren lediglich auf technologieneutralen Modellartefakten beruht, entstehen zur Erzeugung technischer Varianten keine Mehraufwände in der Modellierung. Ein zentrales Modell dient zur Erzeugung multipler technischer Varianten.

Erhöhte Wiederverwendbarkeit der Modelle durch Kombinierbarkeit in variierenden Szenarien

Die datenzentrierte Modellierung der Benutzungsschnittstellen ermöglicht die freie Komposition erstellter Modelle. Wiederverwendete Komponenten können feingranular angepasst werden. Da diese Anpassungen erst zum Zeitpunkt der Wiederverwendung für ein spezifisches Szenario erfolgen, können die Komponenten in beliebige Szenarien eingebunden

werden, die zum Zeitpunkt ihrer Erstellung noch nicht bekannt waren. Im Vergleich zu bestehenden Ansätzen erhöht sich die Wiederverwendbarkeit existierender Anwendungen in der Praxis dadurch wesentlich.

Gemeinschaftliche Nutzbarkeit der Modelle in einem Dienst-Ökosystemen

Die vorgeschlagene Repräsentation des datenzentrierten Modells als annotierte Ontologie ermöglicht die gemeinschaftliche Nutzbarkeit in einer Vielzahl von technischen Umgebungen. Es wird eine universell nutzbare Darstellung erreicht, die eine weitreichende Verteilbarkeit der Modelle ermöglicht und von Dritten interpretiert werden kann. Die zusätzliche Beschreibung der Korrelation erfasster Daten auf Schnittstellendaten von Ökosystem-Diensten ermöglicht die Nutzung erstellter Modelle für multiple Dienst-Implementierungen. Die Repräsentation des Modells besteht dabei aus lediglich einem im Sinne der Generierung vollständigen Artefakt, welches als *in sich geschlossene Einheit* zur Verfügung gestellt wird.

Der dargestellte Architekturansatz für eine *Shared UI*-Infrastruktur bietet schließlich den technischen Rahmen für die gemeinschaftliche Nutzung von Benutzungsschnittstellen. Im dargestellten Ökosystem sind Modelle zur Wiederverwendung frei zugänglich, können in beliebige Benutzungsschnittstellen transformiert und die zur Laufzeit erfassten Informationen für beliebige Dienste aufbereitet werden. Dies schafft die Infrastruktur für eine weitreichende gemeinschaftliche Nutzung.

11.2.2 Neuartigkeit des Ansatzes

Die Ergebnisse der Arbeit liefern neuartige Lösungen für die Problembereiche *Varianten*, *Komposition*, *vollautomatisierte Generierung*, *Wiederverwendung* und *gemeinschaftliche Nutzung*, die von bestehenden Ansätzen nicht oder nur punktuell abgedeckt werden (vgl. Abschnitt 3.2.3). Die Neuerungen liegen insbesondere in folgenden Eigenschaften:

- Fokussierung auf die erfassten Daten einer Anwendung
- Modellierung von Benutzungsschnittstellen in nur einem Artefakt
- Modellierung von Varianten als Differenz zum Ausgangsmodell
- Integration bestehender Komponenten und Methodik zu deren Anpassung
- Vollständig automatisierte Erzeugung von Varianten und Kompositionen
- Universelle Repräsentation des Modells
- Modellierung der Anbindung an Dienste
- Gemeinschaftliche Nutzbarkeit als *Shared UIs*

Insbesondere deren **Zusammenführung in einem durchgängigen Ansatz** stellt eine wesentliche Neuerung gegenüber bestehenden Ansätzen dar.

Die Lösung zur **gemeinschaftlichen Nutzung als *Shared UIs*** führt darüber hinaus zu einer vollständigen Entkopplung der Erstellung von Benutzungsschnittstellen, deren konkreter Ausprägungen und anzubindender Dienste. Da sowohl die Benutzungs- als auch die Dienst-Schnittstellen als *shared knowledge* semantisch spezifiziert und frei verfügbar sind, können frei zugängliche Ökosysteme für Anwendungsdomänen entstehen, die eine weitreichende Wiederverwendung ermöglichen. Die notwendigen Schritte *hin* zu einer finalen Benutzungsschnittstelle und *zurück* zur Verarbeitung der erfassten Daten in Diensten können von unterschiedlichen Anbietern übernommen werden. Dies ist eine **neuartige Form der gemein-**

schaftlichen Nutzung, die auch funktionale Aspekte berücksichtigt und mit bestehenden Ansätzen nicht erreicht werden kann.

Eine Stärke der Lösung ist ihr **modularer Charakter**. Nicht nur die beschriebene, komplexe Aufgabenstellung ist damit umsetzbar, es können bereits Teilaspekte unabhängig voneinander genutzt und kombiniert werden. Die Voraussetzung ist dabei lediglich die datenzentrierte Modellierung, aus der automatisiert Benutzungsschnittstellen generierbar sind. Diese kann bedarfsorientiert um die Aspekte

- Modellierung inhaltlicher Varianten
- Modellierung von Kompositionen
- Generierung multipler technischer Varianten
- Anbindung an Dienste
- Bereitstellung zur gemeinschaftlichen Nutzung
- Aufbau und Nutzung eines Ökosystems

ergänzt werden. Das ermöglicht die Nutzung der Vorteile jedes Aspekts bereits *im Kleinen* (z.B. in einem Projekt) und bleibt dennoch offen für zukünftige Erweiterungen (z.B. sukzessive Ausweitung in eine Unternehmens- oder Ökosystem-Infrastruktur).

Diese durchgängige und dennoch modulare Lösung kann mit bestehenden Ansätzen nicht erreicht werden und ist die **Hauptinnovation des entwickelten Ansatzes**.

11.2.3 Auswirkungen auf die Entwicklung und die Endnutzer

Die in der Arbeit dargestellte Lösung hat nicht nur Auswirkungen darauf, wie dialogbasierte Anwendungen erstellt werden, sie schafft auch einen Mehrwert für die Endnutzer dialogbasierter Anwendungen.

Auswirkungen auf die Entwicklung. Diese liegen zuallererst in der signifikanten Reduktion der Aufwände zur Entwicklung einer Benutzungsschnittstelle. Die datenzentrierte Modellierung verschiebt zudem den Fokus von einer technischen, UI-bezogenen auf eine fachliche Perspektive. Dies kann zu Rollen- und Aufgabenveränderungen in Projekten genutzt werden. Die vergleichsweise einfache Verständlichkeit und Technologieneutralität der Modelle schafft die Option, dass diese weitgehend von *Personen mit geringer technischer Expertise* erstellt werden können (z.B. Beschäftigten eines Fachbereichs). Tiefergehendes technisches Wissen ist jedoch weiterhin zur Schaffung der technischen Voraussetzungen notwendig, insbesondere zur Entwicklung von Geschäftslogik, von Generatoren für neue Plattformen und der Bereitstellung einer technischen Infrastruktur.

Erfolgt die Entwicklung in einem Szenario der *gemeinschaftlichen Nutzung* in einem Benutzungsschnittstellen-Ökosystem, sind die Auswirkungen weitreichender. Hier führt der Ansatz zu einem **neuartigen Paradigma für die Anwendungserstellung**, in welchem unterschiedliche Anbieter *unabhängig voneinander* Teile der Infrastruktur bereitstellen können. Für die verschiedenen Komponenten der Architektur (vgl. Abschnitt 9.3, Abbildung 9.2: *fachliche Dienste*, *Applikations-*/*Variantenmodelle*, *Repository-*, *Transformations-* und *Instanzdatendienste*) entstehen neue, spezialisierte Rollen:

- **Dienst-Anbieter** (z.B. Unternehmen, Institutionen) bieten konkrete Ökosystem-Dienste an, indem Sie für deren standardisierte Schnittstellen Implementierungen bereitstellen. Sie können auf diese Weise ihre Dienstleistungen anbieten, ohne den Aufwand der Integration in Benutzungsschnittstellen zu erbringen.

- **Benutzungsschnittstellen-Anbieter** (z.B. Unternehmen, Institutionen, Nutzer, Communities) erstellen Modelle und Varianten von Benutzungsschnittstellen für standardisierte Ökosystem-Dienste. Die Modelle werden in einem UI-Ökosystem zur gemeinschaftlichen Nutzung angeboten und können mit beliebigen Dienst-Anbietern zusammenspielen.

- **Applikations-Repository-Anbieter** (z.B. Unternehmen, Organisationen, Communities) können die Rolle von *Maklern* übernehmen. Sie können Modelle für spezifische Nutzergruppen oder basierend auf bestimmten Kriterien bündeln (z.B. domänen- oder qualitätsbezogen). Nutzern wird so die Auswahl auf ihre Bedürfnisse zugeschnittener Modelle erleichtert. Zusätzlich können sie die Auswahl passender Dienst-Anbieter unterstützen (z.B. über Bewertungssysteme zur qualitativen Auswahl) und so die Verbindung von Dienst und Benutzungsschnittstelle herstellen.

- **UI-Transformations-Anbieter** bieten spezialisierte Generatoren für konkrete Technologien, Plattformen oder Nutzergruppen an. Sie tragen zur Vielfalt der Benutzungsschnittstellen in einem Dienst-Ökosystem bei, indem sie Nutzern unterschiedlichste UI-Transformationen zur Verfügung stellen, die auf spezifische Nutzergrupppen und Anwendungsbereiche zugeschnittene Benutzungsschnittstellen erzeugen (z.B. sprachbasierte Eingabe, Spezialgeräte, Menschen mit besonderen Anforderungen).

- **Nutzer** eines solchen Ökosystems (z.B. Unternehmen, Institutionen, Portal-Integratoren) können für ihre Anwendungen aus den frei verfügbaren Modellvarianten auswählen, diese kombinieren und mit konkreten Dienst-Implementierungen verbinden. Sie können Anbieter wählen, die aus den Modellen spezifische finale UIs erzeugen, die sich nahtlos in den Kontext ihrer Anwendung integrieren lassen.

Die Auswirkungen liegen in einer **weitreichenden Entkopplung der Aufgaben**, die weit über Projektgrenzen hinausgehen und zu einer Vielfalt an Varianten führen. Die Verantwortung für diese Varianten liegt nicht mehr allein in der Hand der Dienst- bzw. Benutzungsschnittstellen-Entwickler. Vielmehr verteilt sie sich auf viele Rollen, die von Spezialisten übernommen werden. Die Entkopplung kann zudem zu neuen Geschäftsmodellen führen, in denen Anbieter ihre Expertise einer breiten Kundschaft zur Verfügung stellen.

Auswirkungen auf die Endnutzer. In der Vielfalt möglicher Varianten liegen in erster Linie auch die Vorteile für Endnutzer. Einmal erstellte Benutzungsschnittstellen können in Varianten für spezifische Zielgruppen auf deren Bedürfnisse zugeschnitten bereitgestellt werden. Die Generatoren werden von *Spezialisten* bereitgestellt, die diese Anforderungen kennen und berücksichtigen.

Dies kann einerseits zu einer **Qualitätsverbesserung**, andererseits zur **Standardisierung und Vereinheitlichung** spezialisierter Benutzungsschnittstellen führen. Beispielsweise können standardisierte Interaktions- und Navigationsmuster für Nutzer mit Sehbeeinträchtigung umgesetzt werden, die eine einheitliche, anwendungsübergreifende Orientierung in dialogba-

sierten Anwendungen ermöglichen. Transformatoren für diese Standards können von Dienst-Anbietern zur optimierten Unterstützung der Endnutzer *mitgenutzt* werden, ohne Aufwand in deren Entwicklung zu investieren.

Mit dem Ansatz sind noch weitreichendere Freiheitsgrade möglich. Im dargestellten Benutzungsschnittstellen-Ökosystem könnte der **Endnutzer potentiell selbst bestimmen**, welche technische Variante oder von welchem Anbieter **seine persönliche UI** erzeugt werden soll. So könnten z.B. Nutzer mit Sehbeeinträchtigung ihren bevorzugten Anbieter für UI-Transformationen übermitteln (z.B. für Webanwendungen als *Header*-Parameter, der bei jeder Anfrage mitgesendet wird). Dienst-Anbieter werten diese Information aus und delegieren die Erzeugung der UI an den angegebenen bzw. vertrauenswürdige Anbieter[52]. Auf diese Weise erhält der Endnutzer das von ihm persönlich bevorzugte und auf ihn zugeschnittene Ergebnis.

Insbesondere wenn eine Vielzahl von Anbietern partizipiert, kann der Ansatz weitreichende Verbesserungen für Endnutzer bewirken. Da die Bereitstellung von Varianten für den Anbieter eines Dienstes mit geringem bzw. keinem Aufwand verbunden ist, erhöht dies die Wahrscheinlichkeit, dass zukünftig Dienste auch für kleine Zielgruppen mit besonderen Anforderungen angeboten werden.

11.3 Ausblick

Aus den Ergebnissen lassen sich weiterführende Forschungsfragen ableiten, die im Rahmen der Arbeit nicht bearbeitet wurden. Im Folgenden werden daher offene Fragestellungen angeführt, die in weiteren Arbeiten bearbeitet werden müssten (s. Abschnitt 11.3.1).

Zudem lag der Fokus dieser Arbeit auf dienstorientierten Geschäftsanwendungen mit grafischen Benutzungsschnittstellen. Eine Erweiterung des Blickwinkels auf weitere Anwendungsbereiche zeigt, wie dort die Ergebnisse nutzbringend eingesetzt werden könnten (s. Abschnitt 11.3.2).

11.3.1 Offene Forschungsfragen

Die Untersuchung der folgenden Fragen würden die Tragfähigkeit des dargestellten Ansatzes mit quantitativen und qualitativen Ergebnissen ergänzen:

- In welchem Maß verringert sich der Aufwand für Entwicklung und Wartung?
- Sind die bisher gefundenen Interaktionsmuster vollständig?
- Für welche Nutzungskontexte ist der Ansatz nicht geeignet?
- Existieren kritische Szenarien aus technischer Sicht?
- Für welche Domänen sind Ökosysteme sinnvoll nutzbar?

Die folgenden Abschnitte erläutern den potentiellen Erkenntnisgewinn und skizzieren Vorgehensweisen zu deren Beantwortung.

[52] Mit einer solchen Möglichkeit gehen etliche Risiken bezüglich der Anwendungssicherheit einher. Die Einbindung von ungeprüften Anwendungen unbekannter Dritter öffnet *weit* die Tore für eine Vielzahl von Angriffen (z.B. Cross-Site-Scripting- und Cross-Site-Forgery-Attaken), die vor der Umsetzung eines solchen Szenarios intensiv untersucht und gelöst werden müssen.

In welchem Maß verringert sich der Aufwand für Entwicklung und Wartung?

In der Arbeit wurden die konkreten Aufwands- und Kosteneinsparungen **argumentativ**[53] durch die Einfachheit des datenzentrierten Modells, die differentielle Variantenbeschreibung und Wiederverwendbarkeit begründet. Geringe Artefaktzahl, geringe Komplexität und erforderliche Expertise sowie ein höherer Automatisierungsgrad lassen dies offensichtlich erscheinen. Diese Aussagen können in zukünftigen Arbeiten durch empirische Studien mit quantitativen Aussagen detailliert werden.

Die Untersuchung kann z.B. über eine vergleichende Studie erfolgen, in denen konkrete **Entwicklungs- und Wartungsszenarien unterschiedlicher Komplexität von Vergleichsgruppen** durchgeführt werden. Die erste Gruppe erstellt die Szenarien mit herkömmlichen Mitteln (z.B. als webbasierte Anwendung mit Frameworks Assets und Bausteinen, die in einem Referenz-Unternehmen verfügbar sind). Die zweite Gruppe nutzt den vorgestellten Modellierungsansatz unter Nutzung von Generatoren, die entsprechende Assets und Bausteine verwenden. Um das Ergebnis nicht zu verfälschen, ist darauf zu achten, dass die Gruppen einen vergleichbaren Wissensstand besitzen und dass die Assets/Bausteine gleichwertig sind. Hierfür sind im Vorfeld gleichwertige Grundlagen zu schaffen und die Vergleichsgruppen entsprechend zu schulen. Dies liefert bereits erste quantitative Aussagen zum *Rüstaufwand*, der für die jeweilige Vergleichsgruppe zu erbringen ist.

Die untersuchten Szenarien sollten die (1) *Neuerstellung* mehrerer Benutzungsschnittstellen sowie die (2) *Erstellung von Varianten* umfassen. Diese sollten unterschiedliche Komplexität und Dynamik aufweisen. Zudem sollte die (3) *Komposition von Anwendungen* (Wiederverwendung) untersucht werden, soweit sie mit den herkömmlichen Mitteln funktional vergleichbar umsetzbar ist.

Ein weiterer Parameter kann der Kenntnisstand der Mitglieder einer Vergleichsgruppe sein. So können die Gruppen in Untergruppen mit unterschiedlichen Vorkenntnissen aufgeteilt werden, welche o.g. Untersuchungen getrennt voneinander durchführen. Dadurch sind Aussagen zur erforderlichen Expertise und zur technischen Komplexität der Lösung möglich.

Sind die bisher gefundenen Interaktionsmuster vollständig?

Der Kern des Lösungsansatzes liegt in der datenzentrierten Beschreibung einer Benutzungsschnittstelle. Ein Anwendungsmodell muss daher alle notwendigen Informationen zur Generierung gängiger und zukünftiger Interaktionsmuster beinhalten. Ein vollständiger, strukturierter Nachweis der Universalität scheint aufgrund der Vielfalt an Möglichkeiten unmöglich. Der Antwort kann sich jedoch durch sukzessive Untersuchung konkreter Nutzungskontexte angenähert werden.

Im Verlauf der Arbeit erfolgte die Validierung in unstrukturierten *ex-ante*- und *ex-post-Experten-Interviews* für die ausgewählten Anwendungsfälle und kann daher lediglich als Indikator für die Vollständigkeit dienen. Weitere Untersuchungen sollten in Form **strukturierter Experten-Interviews** erfolgen, die bereits *ex-ante* für neue Nutzungskontexte zur Untersuchung der Vollständigkeit durchführbar sind. Diese liefern Antworten zur Vollständigkeit bereits vor der konkreten Umsetzung. *Ex-post* durchgeführte Interviews können anschließend fehlende Aspekte in einer Umsetzung aufdecken, die ggf. zur Erweiterung des Mo-

[53] Validierung als *informed argument* bzw. *logical argument*, vgl. Abschnitt 4.2, Tabelle 4.2

dells führen. Die Experten müssen dabei sowohl das technische (z.B. Web- oder RichClient-Anwendungen) als auch das fachliche Umfeld und die dort genutzten Interaktionsmuster bestehender Anwendungen kennen.

Da die *ex-ante*-Untersuchungen vor einer Umsetzung erfolgen, sind sie hinsichtlich des geringen Aufwands einfach durchführbar. In zukünftigen Arbeiten sind diese für weitere Nutzungskontexte durchzuführen, um die Vollständigkeit für neue Anwendungsbereiche zu indizieren oder Erweiterungsbedarf aufzeigen.

Für welche Nutzungskontexte ist der Ansatz nicht geeignet?

Die Arbeit konzentriert sich bewusst auf dialogbasierte, formularartige Benutzungsschnittstellen. Diese besitzen eine hohe Relevanz im Bereich der Unternehmensanwendungen und weisen einen hohen Standardisierungsgrad auf, der die vollständig automatisierte Erstellung begünstigt. Das Augenmerk lag hauptsächlich auf grafischen Benutzungsschnittstellen, aus welchen die Anforderungen abgeleitet und die Evaluationen durchgeführt wurden. Auf weitere Arten von Benutzungsschnittstellen (z.B. akustisch, haptisch) wurde lediglich ohne systematische Betrachtung verwiesen. Obwohl der praktische Nutzen des Ansatzes in diesen Bereichen gezeigt wurde, können Anwendungskontexte existieren, in denen hinsichtlich der Nutzbarkeit (*Usability*) nur suboptimale oder keine Ergebnisse erzielt werden.

Der Nachweis der Universalität kann aufgrund der Vielzahl an Möglichkeiten auch hier nur durch Annäherung anhand konkreter Fälle erfolgen. Hierzu müssen in zukünftigen Arbeiten gezielt weitere technische Kontexte (z.B. sprachbasierte Schnittstellen, taktile Geräte) sowie spezielle Nutzergruppen untersucht werden. Um zu einer tragfähigen Aussage zu gelangen, kann dies als Umsetzung konkreter Fällen erfolgen, die als Ergänzung zu den o.g. *ex-ante* Experten-Interviews umgesetzt werden. Die zu untersuchenden Fragestellungen sind dabei:

- Ist die Umsetzung für eine Technologie und/oder Nutzergruppen geeignet?
- Können Unzulänglichkeiten bereits mit dem bestehenden Modell gelöst werden?
- Kann das Modell ggf. um fehlende Aspekte ergänzt werden?
- Handelt es sich bei der Umsetzung um eine Technologie oder Nutzergruppe, die mit dem Ansatz unmöglich unterstützt werden kann?

Zur Untersuchung eignen sich folgende Vorgehensweisen: *ex-ante* durchgeführte **Experten-Interviews** liefern die Anforderungen, anhand derer iterativ das konkrete Szenario in **Fallstudien** umgesetzt wird. Zur Untersuchung der *Usability* des Ergebnisses eigenen sich anschließend insbesondere **empirische Nutzertests**, die strukturiert durchgeführt werden und entweder zur Verbesserung der Umsetzung, der Erweiterung des Modells oder als Negativ-Nachweis der Anwendbarkeit des Ansatzes auf den Anwendungsfall dienen.

Existieren kritische Szenarien aus technischer Sicht?

Die Beantwortung dieser Frage liefert quantitative Aussagen zum Lösungsansatz. Die technische Umsetzung der Lösung erfolgte in der Arbeit zwar prototypisch, wird dennoch in Teilen bereits in realen Systemen produktiv eingesetzt. Eine Analyse des Laufzeitverhaltens erfolgte dabei im Zuge der Einführung und führte bereits zu technischen Optimierungen (z.B. Nutzung von Caching-Mechanismen). Auch wenn das Laufzeitverhalten der Lösung in den

betrachteten Fällen unproblematisch war, können weitere Untersuchungen des Gesamtansatzes bzw. einzelner Funktionalitäten um Messergebnisse ergänzen.

Das Ziel zukünftiger Untersuchungen ist dabei die Identifikation und Charakterisierung *kritischer Szenarien* sowie Strategien zu deren Lösung. Ein solches Szenario ist z.B. das Verhalten unter hoher Last bei zur Laufzeit erzeugten, komplexen Kompositionen, deren Komponenten von unterschiedlichen Servern bezogen werden. Hier bestehen insbesondere Latenzprobleme bei der Beschaffung der Anwendungsbeschreibungen und der Komposition an sich. Aus den bisher gesammelten Erfahrungen lassen sich diese mit gängigen Mitteln (z.B. Mirroring-/Caching-Mechanismen) lösen.

Zur Identifikation kritischer Szenarien können **technische Evaluationen** dienen, welche die Skalierbarkeit nachweisen bzw. Beschränkungen aufzeigen. Diese können zudem quantitative Aussagen treffen, in welchem Maße sich die Nutzung einzelner Aspekte der Lösung (z.B. *Variantenerzeugung*, *Bildung von Kompositionen in Varianten*, *Bezug von Komponenten*) auf das Laufzeitverhalten auswirken und so die Grundlage zur Bewertung für zukünftige Einsatzgebiete ermöglichen.

Für welche Domänen sind Ökosysteme sinnvoll nutzbar?

In der Arbeit wurde am Beispiel von *Distributed Marketspaces* exemplarisch gezeigt, dass ein Aufbau von digitalen Ökosystemen und deren Benutzungsschnittstellen grundsätzlich möglich ist. Die Arbeiten von Radonjic-Simic et al. (vgl. [168, 183–185]) zeigen zudem mögliche Anwendungsbereiche auf. Wenngleich die Ergebnisse der Arbeit die grundlegenden Konzepte und Architektur für ein Dienst- und UI-Ökosystem liefern, könnte sich deren organisatorischer Aufbau in der Praxis schwierig gestalten. So existieren in der Industrie und Wirtschaft derzeit nur wenige Standards für domänenspezifische Dienst-Schnittstellen, an denen der Ansatz validiert und ein unternehmensübergreifendes Ökosystem aufgebaut werden kann.

Der vorgestellte Ansatz zur gemeinschaftlichen Nutzung von Benutzungsschnittstellen kann diese ersten Entwicklungen unterstützen. Er birgt ein großes Potential für zukünftige Arbeiten, da er den Bestrebungen zum Aufbau von Dienst-Ökosystemen eine Lösung zur Einbindung der Nutzer anbietet. Für den vorgeschlagenen Ökosystem-Ansatz ergeben sich daraus weitere Fragestellungen, die dem Bereich der Wirtschaftsinformatik zugeordnet werden können.

- Welche fachlichen Domänen eignen sich für den Aufbau von Ökosystemen?
- Haben die Akteure einer Domäne Interesse an dem *share everything*-Ansatz?
- Welche Prozesse einer Domäne eignen sich zur gemeinschaftlichen Nutzung?
- Existieren Querbeziehungen zwischen Domänen, die den Aufbau interagierender Ökosysteme motivieren?

Aus der Perspektive der Arbeit ist die zu untersuchende Fragestellung dann: *Ist die vorgestellte Infrastruktur und Architektur für diese Domänen tragfähig und umsetzbar?* Die damit verbundenen Untersuchungen sind im ersten Schritt mit **Experten-Interviews** und **empirischen Studien durchführbar**. In weiteren Schritten können dann anhand von Fallbeispielen entsprechende Ökosysteme aufgebaut und verprobt werden.

11.3.2 Ausblick auf zukünftige Anwendungsbereiche

Der modulare Aufbau der Lösung erlaubt die Anwendung der gefundenen Lösungen in einer Vielzahl von Kontexten. Er kann als **ergänzende Technologie** in allen Bereichen genutzt werden, die einerseits eine Benutzungsschnittstelle für die dialogbasierte Verarbeitung von Daten erfordern, andererseits eine Vielzahl an Personengruppen einbeziehen und dabei unterschiedliche Interaktionsformen berücksichtigen müssen. Im Folgenden wird zunächst der Einsatz als ergänzende Technologie zur Bereitstellung von Benutzungsschnittstellen für folgende Anwendungskategorien bzw. Basistechnologien skizziert:

- Software-Produktlinien
- Workflow-gestützte Anwendungen

Anschließend werden anhand folgender Beispiele konkrete, fachliche Anwendungsbereiche aufgezeigt:

- Automotive-Anwendungen
- Internet of Things
- Digitalisierung im öffentlichen Bereich

Sie sollen den Nutzen und die Vorteile des generativen Ansatzes in diesen Bereichen illustrieren, die sich auf viele Felder übertragen lassen.

UIs für Software-Produktlininen

Der Ansatz von Software-Produktlinien (SPL) [39, 154, 174] besteht darin, nutzungsspezifische Varianten von Anwendungen aus einer Menge vorgefertigter Software-Komponenten mit technischen Mitteln und Verfahren zu erstellen.

> *"A **software product line** is a set of software-intensive systems that share a **common, managed set of features** satisfying the specific needs of a particular market segment or mission and that are developed from a common **set of core assets** in a **prescribed way**."* [154]

Hierzu werden vorgefertigte Software-Bausteine (SPL-Module) basierend auf festgelegten Rahmenbedingungen (SP-Spezifikation) kombiniert und zu einer Produktvariante zusammengefügt. Die SP-Spezifikation für eine Produktvariante beinhaltet Vorgaben, welche SPL-Module und funktionale *Features* in das Endprodukt einfließen sollen und wie diese variieren.

Insbesondere für mit SPL-Verfahren zu erzeugende dialogbasierte Produktvarianten können die Lösungen der Arbeit zur Generierung benötigter UIs genutzt werden. Hierzu kann für ein SPL-Modul anhand der zu erfassenden Daten ein datenzentriertes Benutzungsschnittstellenmodell erstellt und als Teil des SPL-Moduls als *asset* bereitgestellt werden. Im Rahmen des SPL-Verfahrens wird anhand des Modells eine zur Produktvariante passende UI-Variante für das SPL-Modul erstellt, indem eine vorgefertigte Variantenbeschreibung genutzt oder aus der SP-Spezifikation abgeleitet wird. Die so erzeugten UI-Varianten der SPL-Module werden zu einer UI kombiniert, sowie die erforderlichen Anpassungen für deren Zusammenspiel in der Produktvariante vorgenommen.

Der Mehrwert bei Nutzung des vorgestellten Ansatzes besteht einerseits in einer einheitlichen Modellierung kombinier- und variierbarer UIs für SPL-Module, andererseits im Abstraktionsgrad der modellierten Benutzungsschnittstelle, der die Bereitstellung einer Produktvariante in verschiedenen technischen Umgebungen ermöglicht. Das Vorgehen zur Variantenbildung und Generierung kann in SPL-Verfahren integriert werden und stellt daher eine Ergänzung bestehender Ansätze dar.

UIs für Workflow-gestützte Anwendungen

Vereinfacht zusammengefasst steuern Workflow-Systeme komplexe, ggf. zeitlich entkoppelte Vorgänge zur Durchführung von Prozessen. Die Prozesse werden dabei in einzelne Schritte gegliedert, die abhängig vom Zustand (z.B. eingegebene und ermittelte Daten) durchgeführt werden. Prozessschritte können dabei entweder parallelisiert oder sequentiell ausgeführt werden. Ein Prozessschritt kann dabei autonom arbeiten oder erfordert zur Durchführung Nutzereingaben (vgl. [111]). Ein Prozess wird als Workflow modelliert, der die ggf. bedingte Abfolge, Parallelisierbarkeit von Schritten sowie Aktionen und beteiligte Daten beschreibt. Für Prozessschritte, die eine Nutzereingabe erfordern, muss zudem eine entsprechende Benutzungsschnittstelle bereitgestellt werden.

Die Ergebnisse der Arbeit können zur Bereitstellung der Benutzungsschnittstellen für Prozessschritte mit Nutzereingaben verwendet werden. Mit einem Prozessschritt wird hierfür eine UI-Beschreibung assoziiert. Bei Aktivierung des Prozessschritts wird daraus eine finale UI mit dem angegebenen Verfahren erzeugt und angezeigt. Die von der UI erfassten Daten dienen dann als Eingabe für den Prozessschritt, woraufhin das Workflow-System mit der Bestimmung von Folgeschritten reagieren kann. Lassen sich Prozessschritte parallelisieren, könnten deren UIs als Komposition dargestellt werden. Ist der Prozess für unterschiedliche Nutzergruppen konzipiert, können zudem Varianten der Benutzungsschnittstelle erzeugt werden.

Der Mehrwert bei der Nutzung des vorgestellten Ansatzes liegt in der Bereitstellung nichttrivialer Benutzungsschnittstellen in nutzerspezifischen Varianten für Prozesse. Da bestehende Workflow-Systeme meist nur rudimentäre Mechanismen zur automatischen Erzeugung von UIs bieten, müssen für komplexere Anwendungen Benutzungsschnittstellen häufig manuell erstellt werden. Bei Nutzung der vorgestellten Lösung ist dies nicht erforderlich, da die mit den Schritten assoziierten UIs zur Erzeugung von Varianten geeignet sind. Mit den Lösungen zur Erzeugung inhaltlicher und technischer Varianten werden Workflow-Prozesse in variierenden Umgebungen und für unterschiedliche Nutzergruppen verfügbar. Workflows werden darüber hinaus in einem Ökosystem-Szenario nutzbar, wie dies im Rahmen der Arbeit für Dienste beschrieben wurde.

UIs im Automotive-Bereich

Die Ergebnisse der Arbeit können zur Erzeugung von Benutzungsschnittstellen-Varianten für Informationssysteme im Automotive-Bereich genutzt werden. In diesem Umfeld werden bereits heute bei einigen Herstellern generative Ansätze genutzt (vgl. Meixner et al. [139]). In diesem Anwendungsfall sind inhaltliche Varianten von UIs interessant, die sich z.B. aufgrund eines Fahrzeug-Modells, einer Fahrzeug-Modellvariante, Ausstattungs-Paketen oder dem Fahrertyp unterscheiden. Neben den inhaltlichen variieren auch die technischen Rahmenbedingungen. So werden z.B. modellabhängig unterschiedliche Displaygrößen und Ein-

gabegeräte eingesetzt, die variierende Benutzungsschnittstellen erfordern. Die Kommunikation der Benutzungsschnittstelle mit den Fahrzeugkomponenten erfolgt u.a. über standardisierte Schnittstellen (z.B. CAN-Bus). Abstrahiert sind diese mit den im Rahmen der Arbeit beschriebenen Diensten vergleichbar und bilden eine standardisierbare Basis, auf denen Benutzungsschnittstellen arbeiten können.

Die Ergebnisse der Arbeit sind daher direkt auf die Erzeugung von UI-Varianten in diesem Anwendungsfall übertragbar. Dafür würden datenzentrierte UI-Beschreibungen aller Anwendungen und optionaler Komponenten erstellt, die für eine Fahrzeugplattform relevant sind. Anhand von Variantenbeschreibungen können daraus modellspezifische UIs erzeugt werden, die zudem die UIs von ausstattungsspezifischen Komponenten aggregieren. Da im Automotive-Bereich die Rechnerkapazitäten eingeschränkt sind, bietet sich hier die Generierung der Varianten zum Entwicklungszeitpunkt an.

Ein Mehrwert des vorgestellten Ansatzes liegt in der abstrakten, datenzentrierten Ausgangsbeschreibung. Hiermit könnten die UIs und die Anbindung an standardisierte Fahrzeugkomponenten modellunabhängig beschrieben werden. Die Erzeugung der finalen UI kann modellspezifisch erfolgen. Zudem kann dies zur schnellen Entwicklung und Verprobung von Prototypen für neue Interaktionsformen genutzt werden.

Management von *Dingen* im *Internet of Things*

Ein großes Potential zur Nutzung des vorgestellten Ansatzes liegt im Anwendungsbereich *Internet of Things* (IoT), welches in den vergangenen Jahren sowohl im industriellen als auch privaten Bereich steigende Relevanz erhält. IoT ist ein System-Konzept, in dem technische *Dinge* einen Verbund bilden, autonom Informationen austauschen und zusammenarbeiten.

> *"IoT is a world of interconnected things which are capable of sensing, actuating and communicating among themselves and with the environment (i.e., smart things or smart objects) while providing the ability to share information and act in parts autonomously to real/physical world events and by triggering processes and creating services with or without direct human intervention."* [88]

Im industriellen Bereich (vgl. [113]) wird das System-Konzept z.B. in den Bereichen Stromnetze, Kraftwerke, öffentlicher Nahverkehr, Windräder und der Überwachung und Verwaltung industrieller Ausrüstungsgegenstände (z.B. Sensoren, Werkzeuge und Maschinen) umgesetzt. Auch im privaten Umfeld finden sich inzwischen viele Sensoren und Aktoren (z.B. Temperatur-Sensoren, Beleuchtung, Haushaltsgeräte, HiFi-Anlagen), die in das Heimnetz eingebunden werden. Die *Dinge* existieren ggf. in herstellerspezifischen Varianten und Versionen und kommunizieren über standardisierte Protokolle. Die ausgetauschten Daten können dabei entweder proprietär (z.B. Telemetriedaten eines Bagger-Herstellers) oder standardisiert sein (z.B. Temperaturdaten eines Sensors).

Trotz der Autonomie der Geräte muß der Mensch zeitweilig mitwirken. Die Aufgabenstellungen liegen hier insbesondere in der *Konfiguration, Verwaltung, Überwachung und Wartung* der einzelnen *Dinge* oder eines Verbunds. Hierzu ist der Zugriff auf Einzelgeräte oder deren gesendete Daten zur Zustandsüberwachung erforderlich. Es werden Benutzungsschnittstellen benötigt, die abhängig vom Typ des *Dings* Daten darstellen bzw. die modellspezifische Konfiguration ermöglichen. Die UIs müssen ggf. in einer Benutzungsschnittstelle für den Ge-

räteverbund integriert werden (z.B. ein Verwaltungs-Portal für die Ausrüstungsgegenstände eines Fuhrparks oder die Sensoren/Aktoren eines Heimnetzes).

Der vorgestellte Ansatz kann zum Aufbau eines einheitlichen Rahmens für die Benutzungsschnittstellen zur Verwaltung genutzt werden. Hierzu können *Dinge* die abstrakten UI-Beschreibungen zur Konfiguration und Anzeige ausliefern, die dann mit den vorgestellten Mitteln zu einer auf eine Nutzergruppe zugeschnittenen UI aggregierbar sind (z.B. als Bausteine einer Verwaltungs- oder Wartungsoberfläche). In einem Ökosystem-Szenario können darüber hinaus UI-Beschreibungen für *Dinge* und Modellvarianten dezentral bereitgestellt werden. Die *Dinge* liefern dann Typ- und Modellinformationen oder Referenzen auf modellspezifische UI-Beschreibungen. Ein Mehrwert liegt hier in der Entkopplung des *Dings* von den UIs zur Konfiguration und Anzeige. *Dinge* können ohne Kenntnis der modellspezifischen Details im Verbund integriert und verwaltet werden. Darüber hinaus wird die Aktualisierung von *Dingen* vereinfacht, da deren UIs unabhängig vom Gerät bereitgestellt und aktualisiert werden können.

Digitalisierung von Prozessen im öffentlichen Bereich

Mit der *Digitalen Agenda 2014-2017* beschloss das Bundeskabinett 2014 Maßnahmen zur digitalen Transformation von Prozessen im öffentlichen Bereich und schuf z.B. mit dem Programm *Digitale Verwaltung 2020* Rahmenbedingungen für die "Verwaltung der Zukunft" [22]. Hieraus ergeben sich vielfältige Aufgaben zur Digitalisierung von Prozessen in Bund, Ländern und Gemeinden (vgl. [92]). Die Vision ist, dass Informations-, Kommunikations- und Transaktionsprozesse zwischen Politik, Verwaltung, Bürgern und der Wirtschaft von jedem Ort, zu jeder Zeit und mit jedem Medium erfolgen können, und zwar schnell, einfach, sicher und kostengünstig [87].

In [22] werden Rahmenbedingungen zur Umsetzung genannt. Ein Aspekt ist, zukünftig die öffentliche Verwaltung für die Bürgerinnen und Bürger elektronisch erreichbar zu machen. Die Barrierefreiheit eines digitalen Verwaltungsverfahrens ist dabei eine grundlegende Anforderung. Für eine medienbruchfreie und effiziente elektronische Abwicklung von Verwaltungsleistungen wird zudem eine Standardisierung von Prozessabläufen und Formularen sowie eine fachübergreifende Anwendung gemeinsamer Standards bei deren Beschreibung und Erstellung angestrebt. Die erforderlichen Standardisierungsmaßnahmen sollen durch eine Bundesredaktion mit hoher Methodenkompetenz begleitet werden. Ein Ziel ist auch der Aufbau einheitlicher IT-Infrastrukturen.

Im Zuge der Transformation wurden bereits Konzepte und Themenfelder bearbeitet und in Teilen umgesetzt (z.B. *E-Government, Smart City*). So bieten einige Institutionen (z.B. Behörden, Bürgerämter) digitale Self-Services für Verwaltungsprozesse an. Beispiele sind Bürgerämter, welche die Ummeldung, Anträge für Parkausweise, Führungszeugnisse oder Führerscheine online ermöglichen. Auch im Hochschulbereich werden vermehrt Dienstleistungen digital angeboten, z.B. Anmeldung zu Bachelor-/Master-/Doktoranden-Studiengängen, Anmeldung zu Kursen oder Unterlagen-Einreichung. Dennoch stellt die Transformation Institutionen vor große finanzielle und technische Herausforderungen [23]. Grundlegende technische Herausforderungen sind dabei

- Heterogenität der Prozesse einzelner Institutionen
- Heterogenität der existierenden Systeme in Ländern, Kommunen und Institutionen

- Barrierefreier Zugang für eine Vielzahl an Nutzergruppen mit spezifischen Bedürfnissen und die damit verbundenen Mehrkosten zur Erstellung adäquater Benutzungsschnittstellen

Der in der Arbeit vorgeschlagene Ansatz adressiert diese Herausforderungen und kann die Digitalisierung im öffentlichen Bereich weitreichend unterstützen. Die angebotenen Dienstleistungen stellen Dienste nach dem in der Arbeit beschriebenen Muster dar. Auch wenn sich die verarbeitenden Systeme (i.S. mehrerer Dienst-Anbieter) unterscheiden, so sollen nach [22] zukünftig die Daten, Prozesse und IT-Infrastruktur weitreichend standardisiert werden. Der Aufbau von Dienst-Ökosystemen mit einheitlichen Schnittstellen könnte das Resultat dieser Digitalisierungsbestrebungen sein - sei dies auf Landes- oder Bundesebene.

Die Erstellung von Benutzungsschnittstellen kann unter diesen Voraussetzungen zentral erfolgen und die Ergebnisse von Institutionen gemeinschaftlich genutzt, angepasst und in deren spezifische Umgebung integriert werden. Die Institutionen können weiterhin ihre eigenen Dienst-Implementierungen nutzen. Die Kosten und das Know-How zur Erstellung der Benutzungsschnittstellen und notwendiger technischer Varianten kann verteilt und minimiert werden. Insbesondere der barrierefreie Zugang für viele Bürger wäre so ein mit einfachen Mitteln erreichbares Ziel.

11.4 Fazit

In der Arbeit konnte gezeigt werden, dass die automatische Generierung von Benutzungsschnittstellen und deren Varianten in dienstorientierten Umgebungen möglich und sinnvoll ist. Die Ergebnisse der Arbeit tragen zur Vereinfachung der Erstellung von Benutzungsschnittstellen für eine Vielzahl von Nutzergruppen und technischer Kontexte bei. Sie schaffen eine Umgebung, in der eine Vielzahl von Anbietern ihre Expertise einbringen und gemeinschaftlich Anwendungen für Menschen mit unterschiedlichen Anforderungen und Bedürfnissen erzeugen können.

Meine persönliche Hoffnung ist, das die Ergebnisse der Arbeit Impulse für zukünftige Methoden zur **Anwendungserstellung für die Vielfalt** geben. Die Lösungsvorschläge bilden hierfür eine mögliche Grundlage und bieten eine Chance, viele bisher als Nischengruppen angesehene Nutzergruppen bedarfsgerecht zu unterstützen und an der Digitalisierung teilhaben zu lassen.

A. Anhänge

A.1 Datengrundlage zur Durchführung der Analyse

Die Analyse wurde in Zusammenarbeit mit der Allianz Deutschland AG durchgeführt, die Anwendungen aus fachlichen Kernbereichen im Kunden- und Vertriebsumfeld zur Untersuchung zur Verfügung stellte. Hierbei handelte es sich um im praktischen Einsatz befindliche Anwendungen, die zudem betriebliche Relevanz besitzen. Die Analyse wurde Ende 2012 zu Beginn der Arbeit initial durchgeführt und in Bereichen, in denen sich in den folgenden Jahren signifikante Änderungen ergaben, aktualisiert.

An der Analyse beteiligt waren Produktverantwortliche, Architekten und Entwickler aus den betreuenden IT-Abteilungen der Allianz, die mit ihrer Expertise bei der Informationsbeschaffung und Bewertung von Ergebnissen mitwirkten.

Im Folgenden werden die Schritte der Untersuchungen zusammengefasst, Mengengerüste dargestellt und die Auswahl der Anwendungen begründet, die als Repräsentanten in die detaillierte Analyse der Eigenschaften dialogbasierter Anwendungen in Kapitel 5 eingeflossen sind. Die Auswahl erfolgte in einzelnen Schritten, die im Folgenden beschrieben werden:

- Identifikation geeigneter Prozessbereiche im Unternehmen
- Bestimmung geeigneter/relevanter Anwendungstypen und Repräsentanten
- Auswahl betrachteter Technologien und Plattformen

A.1.1 Identifikation geeigneter Prozessbereiche im Unternehmen

In Versicherungsunternehmen existieren unterschiedlichste Bereiche, die zur Durchführung ihrer Aufgaben Zugriff auf IT-Systeme und IT-gestützte Prozesse benötigen und mehr oder minder kundennahe Prozesse durchführen. Als *kundenferner* sind dabei Bereiche und Prozesse zu sehen, die der Unternehmensverwaltung oder Abwicklung dienen (z.B. interne Lohnbuchhaltung oder In-/Exkasso von Kundenzahlungen). *Kundennahe* Bereiche sind z.B. der Innendienst, das Call- bzw. Service-Center und Vermittler/Makler, die direkten Kontakt zu Kunden pflegen und Tätigkeiten in deren Sinne durchführen. Hinzu kommt der Kunde, der digitalisierte Prozesse über das Internet selbst abwickelt. Insbesondere die **kundennahen Bereiche wurden als sinnvoll für die Generierung von Benutzungsschnittstellen und deren Wiederverwendung** eingestuft, da hier Übereinstimmungen in den Prozessen bestehen und unterschiedliche Technologien zum Einsatz kommen. In den kundenferneren Bereichen sind die Anwendungen meist individuell und existieren genau einmal, wodurch hier die Generierung weniger nutzbringend ist.

Daher wurden folgende Aufgabenbereiche im Unternehmen und damit verbundene Zugänge zu den IT-Systemen wurden daher näher betrachtet:

- Sachbearbeiter-Arbeitsplatz
- Service-Center-Arbeitsplatz
- Vermittler-Arbeitsplatz und -Anwendungen
- Anwendungen im Online-Kundenportal

Für diese Aufgabenbereiche wurden die Arbeitsweisen und vorhandene Benutzungsschnittstellen charakterisiert und darauf aufbauend die potentielle Eignung zur automatischen Generierung und Wiederverwendung bewertet.

Sachbearbeiter-Arbeitsplatz. Eine Aufgabe von Sachbearbeiter im Innendienst besteht in der schnellen Verarbeitung von vertragsbezogenen Kundendaten, die z.B. per Post eingegangen sind (Massenverarbeitung). Eine weitere Aufgabe ist die Bearbeitung von aufwendigeren Anfragen und Klärung von vertragsbezogenen Sachverhalten durch Spezialisten. Die Anwendungsfälle erfordern nur wenig Kundeninteraktion, sind sehr spezifisch und Fallbezogen.

Für die Massenverarbeitung sind die Benutzungsschnittstellen in hohem Maße technischer Natur. Es dominieren hier auch heute noch Host-basierte, textuelle Eingabemasken, die über Terminal-Emulationen angesprochen werden. Diese sind hochgradig spezialisiert und z.T. für Außenstehende nicht verständlich (vgl. Abbildung A.1). Die Spezialisierung geht soweit, dass die Masken auf *"Blindeingabe"* optimiert sind und ggf. Tastatureingaben gepuffert werden, sofern der Erfassende schneller Daten erfasst als das IT-System sie verarbeiten kann. Dies wirkt sich sowohl auf die Struktur als auch auf das Erscheinungsbild der Benutzungsschnittstellen aus.

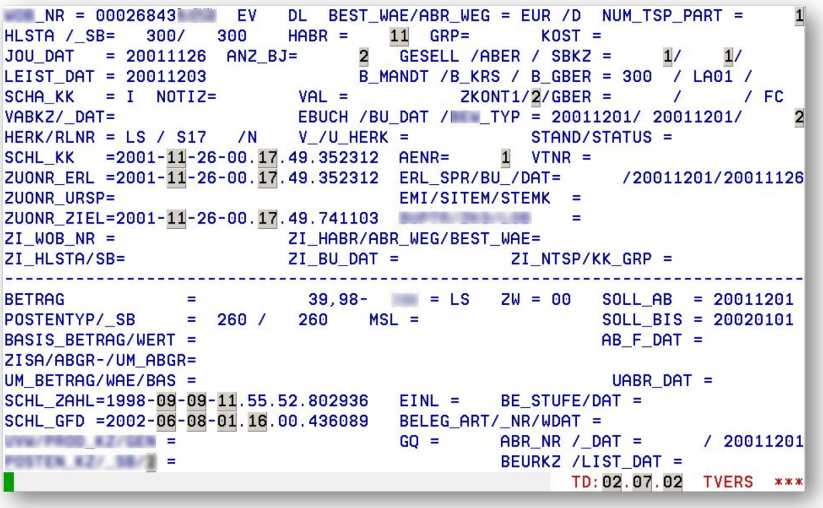


Abbildung A.1. Sachbearbeiter-Arbeitsplatz (Host-Maske)

Für die Bearbeitung der Kundendaten existiert zudem eine Innendienst-Anwendung, die darauf ausgelegt ist, die gesamten Kundendaten in einer Weise zu präsentieren, dass alle Aspekte der Kunden-Vertragsbeziehung bearbeitet werden können und häufig genutzte Informationen effizient erreichbar sind. Die Benutzungsschnittstellen sind daher wenig dialogbasiert i.S. der Arbeit, sondern aggregieren Daten aus der Backend-Datenhaltung auf Masken weitgehend themenbezogen. Sie ermöglichen die Navigation zwischen den Themenbereichen (s. Abbildung A.2), wobei Änderungen an den Daten lediglich punktuell erfolgen. Da es sich um Spezialisten handelt, müssen diese wenig geführt werden; Unterstützungen wie Hilfetexte können entfallen.

Der Sachbearbeiter-Arbeitsplatz bietet **nur geringes Potential zur Generierung und Wiederverwendung** von Benutzungsschnittstellen, da die Masken in Struktur und Aufbau weitgehend spezialisiert sind und punktuelle Änderungen der Daten erfordern. Es finden sich nur

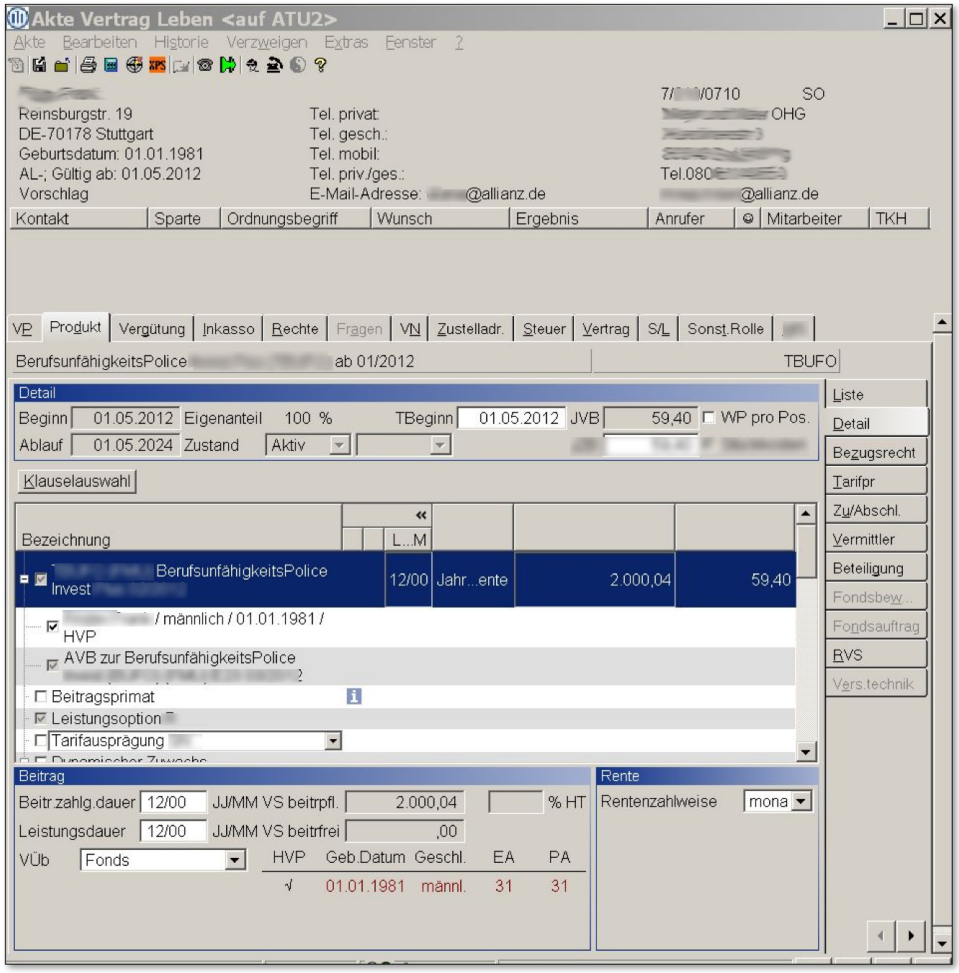


Abbildung A.2. Sachbearbeiter-Arbeitsplatz (Innendienst-Anwendung)

wenig wiederverwendbare Dialoge oder geführte Dialogabläufe. Aus diesem Grund wurde dieser Bereich in der weiteren Analyse nicht betrachtet.

Service-Center-Arbeitsplatz. Dieser Arbeitsplatz ähnelt dem des Innendienstes, besitzt aber eine eher informative Sicht auf den Kunden. Hier steht nicht die Änderung vertragsbezogener Aspekte im Vordergrund, sondern die Verwaltung der Kundenbeziehung (z.B. Ändern von Zugangsdaten, Aktivierung/Deaktivierung von Zugängen, aber auch Informationen zu Verarbeitungsständen). Änderungen betreffen dabei nicht die Vertragsdaten sondern eher organisatorische Aspekte (Ändern von Kommunikationsdaten, Online-Zugängen). Analog dem Innendienst-Arbeitsplatz werden in der Benutzungsschnittstelle Kundendaten aus den Datenbanken aggregiert dargestellt. Auch hier erfolgen Änderungen von Daten in nur wenigen Fällen in geführten Dialogen, sondern werden meist punktuell in den Masken direkt durchgeführt. Es ergibt sich daher analog dem Innendienst-Arbeitsplatz **wenig Potential für Wiederverwendung und Generierung**. Aus diesem Grund wurde dieser Bereich ebenfalls von der weiteren Analyse ausgenommen.

Vermittler-Arbeitsplatz und Anwendungen. Vermittler verwalten die Kundenbeziehungen und generieren ggf. Neugeschäft durch Verkauf von Produkten. Einerseits beraten und informieren Vermittler Kunden bezüglich ihrer bestehenden Verträge, andererseits vermit-

teln sie Neuverträge und beraten bei der Produktwahl. Es existieren unterschiedliche Arten von Vermittlern, die sich im Umfang der Kundendienstleistungen unterscheiden:

- Allianz Ausschließlichkeitsorganisation *AO* (Allianz-Vertreter)
- Allianz Leben Makler (Drittunternehmen beschränkt auf Leben-Produkte)
- Generelle Produkt-Makler (Drittunternehmen, die u.a. Allianz Produkte vertreiben)
- Kooperationspartner (z.B. Unternehmen im Firmenkunden-Geschäft)

Hierbei besitzt die *AO* die engste Bindung an das Unternehmen und benötigt den Zugriff auf das größte Spektrum an Prozessen. Die anderen Vermittlerarten sind eingeschränkter. Immer jedoch ist aus Allianz-Sicht der Vermittler der Repräsentant eines Kunden. Aus Sicht des Kunden ist er der Repräsentant des Unternehmens. Damit gleichen die Geschäftsprozesse des Vermittlers grundsätzlich denen, die Kunden selbst ausführen können, beinhalten aber auch interne Prozesse und eine *Unternehmenssicht* des Kunden.

Für Vermittler existieren mehrere Zugangswege zu Prozessen der Allianz. Vorrangig ist dies der Vermittler-Arbeitsplatz in Form von Rich-Client-Anwendungen auf Laptops bzw. Standrechnern und ein Online-Portal[54] im Extranet des Unternehmens. Auf diese Zugänge verteilen sich die Anwendungen. In beiden Fällen wurde eine Rahmenanwendung geschaffen, in welcher einerseits Kundeninformationen im Überblick dargestellt und aufgaben- und produktspezifische Bausteine von Fachbereichen zur Durchführung deren Geschäftsprozesse integriert werden. In Abbildung A.3 ist die Antragstrecke für eine Lebensversicherung als Desktop-Anwendung dargestellt. Abbildung A.4 zeigt den enthaltenen Bereich der *Risikoprüfung* der gleichen Antragstrecke, allerdings in der technischen Variante der Anwendung im Online-Portal für den Vertreter.

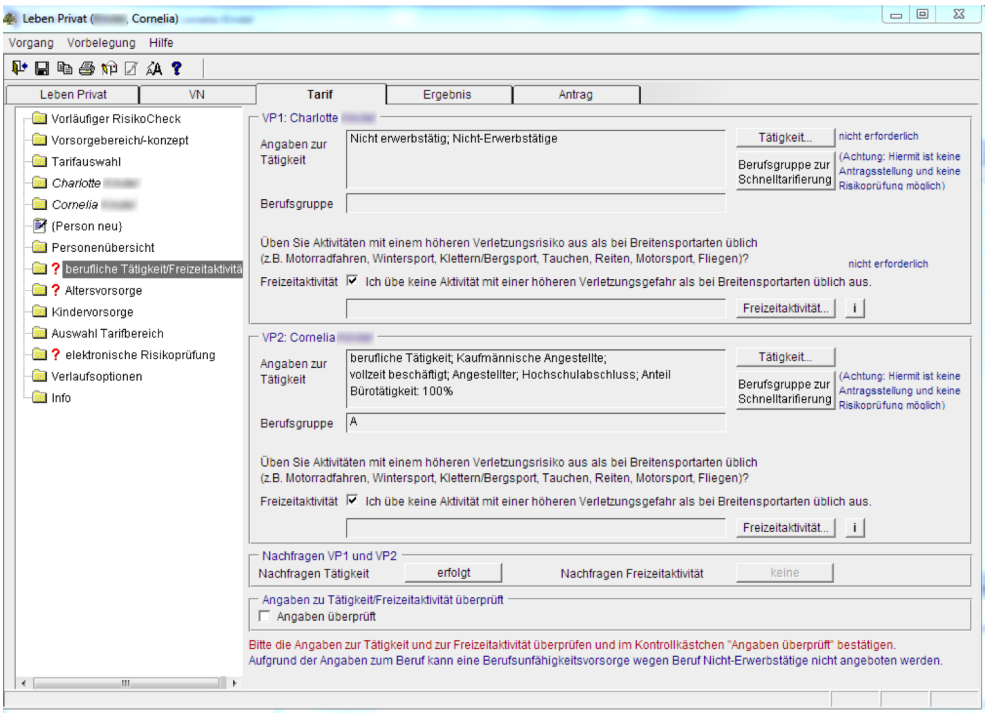


Abbildung A.3. Vermittler-Arbeitsplatz (Desktop Antragstrecke)

[54] Aktuell ist eine Plattform im Aufbau, in welcher diese Zugänge zusammengeführt und hinsichtlich einer optimierten Vertriebsunterstützung funktional erweitert werden.

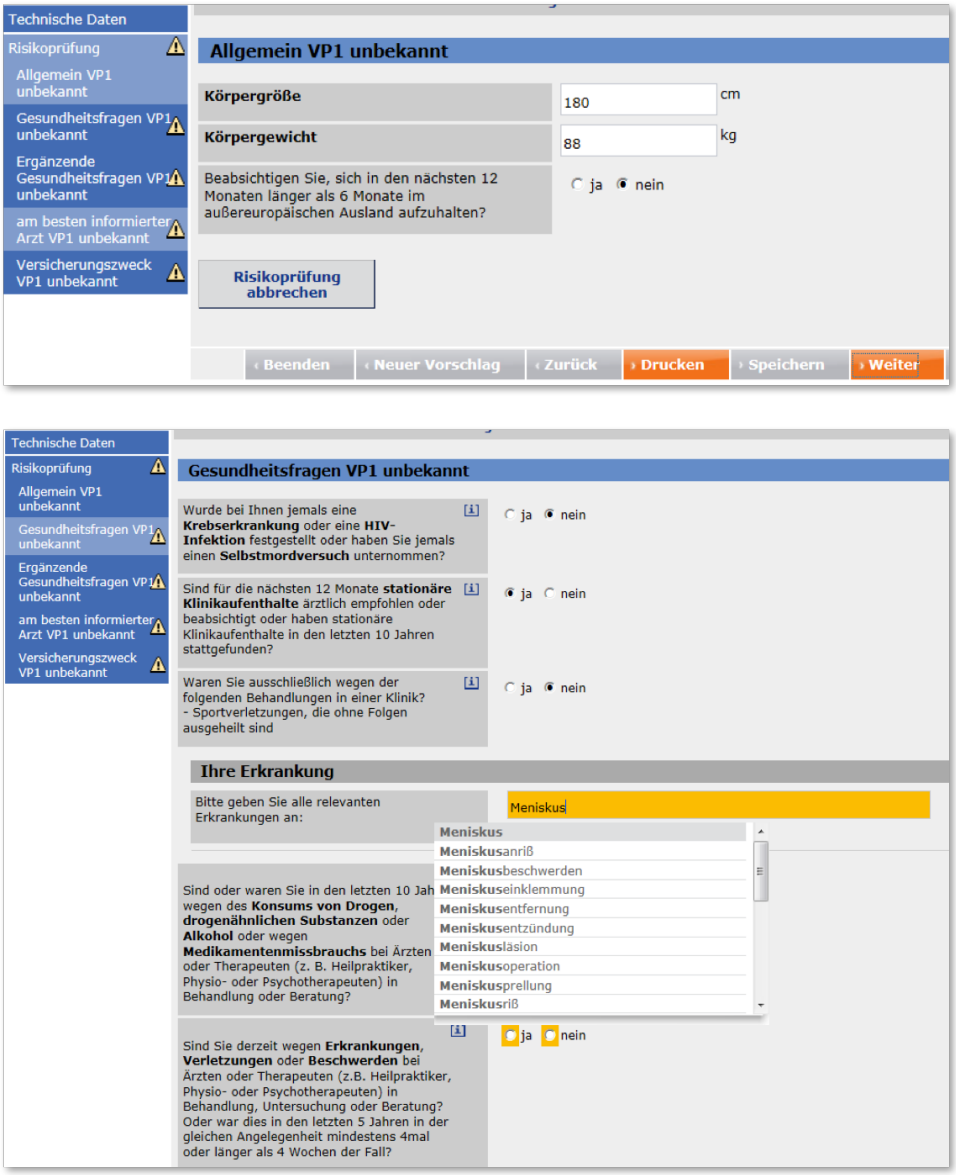


Abbildung A.4. Vermittler-Arbeitsplatz (Online-Portal Antragstrecke)

Aufgrund der Kundennähe des Vermittlers und seiner Repräsentantenfunktion entsprechen die durchgeführten Prozesse denen für Kunden. Insbesondere handelt es sich dabei um Änderungen an bestehenden Vertragsbeziehungen (z.B. Änderung laufender Verträge, Kundendaten, Lebenssituation) und dem Aufbau einer neuen Beziehung (z.B. Antrag auf ein Produkt). Diese Aufgaben werden in den IT-Anwendungen vorwiegend dialogbasiert abgewickelt. Die Komplexität der Anwendungen variiert hierbei von einfachen Änderungen der Stammdaten eines Kunden bis hin zur komplexen Tarifierung eines Angebots für ein Versicherungsprodukt. Die Dialogstrecken verhalten sich dennoch sehr ähnlich und weisen Wiederholungen auf. Diese Eigenschaften weisen darauf hin, dass hier eine automatisierte Erzeugung sowie Wiederverwendung sinnvoll genutzt werden können.

Anwendungen im Online-Kundenportal. Im Kundenportal der Allianz und der Allianz-App kann der Endkunde vermehrt Prozesse durchführen, die seine Beziehung zur Allianz

betreffen, ohne seinen Vermittler zu kontaktieren oder einen Anruf zu tätigen. Einerseits sind dies Prozesse zur Darstellung von Informationen zu Verträgen, andererseits Änderungen an bestehenden Vertragsbeziehungen oder dem Aufbau einer neuen Beziehung - wie sie auch vom Vermittler durchgeführt werden. Hierfür werden im Online-Portal Anwendungen bereitgestellt, die den Kunden dialogbasiert durch den Prozess leiten. Diese dialogbasierten Anwendungen unterliegen Regeln für das Design und Verhalten und weisen sich wiederholende Elemente auf, die auf eine Generier- und Wiederverwendbarkeit deuten.

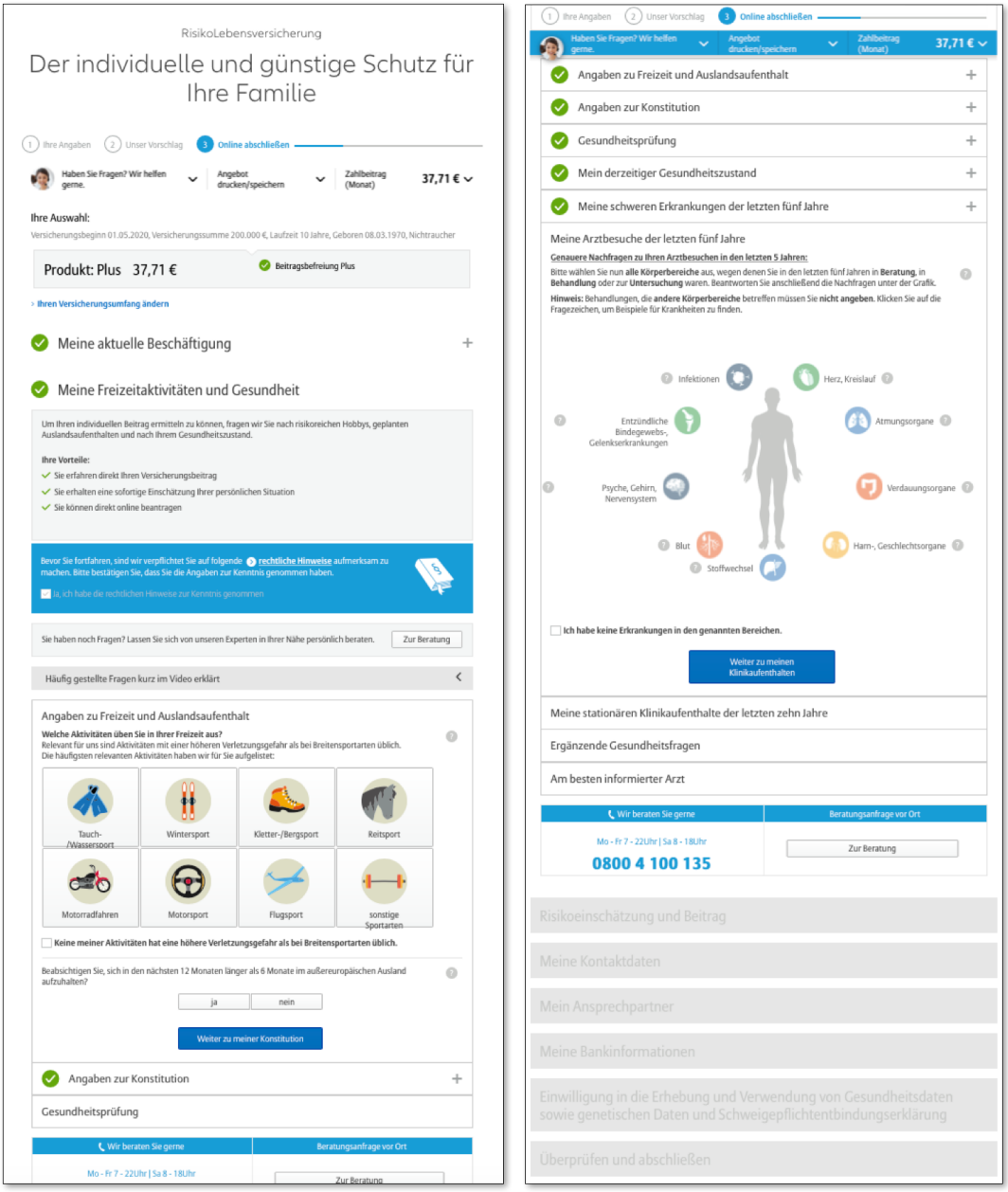


Abbildung A.5. Kunden-Portal (Antragstrecke RLV, Risikofragen)

Neben einfachen Self-Service-Dialogen (z.B. die Mitteilung einer Adressänderung) werden vermehrt auch komplexere Prozesse angeboten, die z.T. umfangreiche Interaktionen mit Backendsystemen beinhalten bzw. Anwendungen miteinander kombinieren (z.B. die Angebotserstellung für eine Lebensversicherung mit Risikoprüfung). In Abbildung A.5 ist exempla-

risch die Einbettung der Risikofragen in den Risiko-Lebensversicherungsantrag (RLV) dargestellt. Sie zeigt einen dialogbasierten Prozess (RLV-Antrag), den es so auch im Vermittler-Arbeitsplatz gibt. Die Darstellung und Aufbereitung (z.B. Informationsgehalt der Seiten) unterscheidet sich jedoch grundlegend.

Ergebnis:

Für die Analyse wurden als repräsentative Nutzergruppen *Vermittler* und *Endkunden* ausgewählt, da sie das Komplexitätsspektrum dialogbasierter Anwendungen am weitesten abdecken. Im Gegensatz zu den sehr spezialisierten Sachbearbeiter- und Service-Center-Anwendungen, eignen sich hier die Benutzungsschnittstellen durch den hohen Standardisierungsgrad und die potentielle Wiederverwendbarkeit ganzer Prozessabläufe besonders zur automatischen Generierung. Die Bereiche weisen zudem funktionale Überschneidungen auf, woraus sich der Nutzen von Varianten für unterschiedliche Nutzergruppen und Plattformen untersuchen lässt.

Die Komplexität der Anwendungen in diesen Gruppen kann als Querschnitt über kundennahe Anwendungen gesehen werden. Während Vermittler-Anwendungen den höchsten Detaillierungsgrad bei der Erfassung von Daten aufweisen, ist dieser bei Endkunden am geringsten. Im Gegenzug erfordern Endkunden-Anwendungen eine striktere Nutzerführung und intuitivere Benutzungsschnittstellen, da weniger Fachwissen vorausgesetzt werden kann, das bei der Bedienung der Anwendung unterstützt (siehe hierzu die Analyseergebnisse in Kapitel 5).

A.1.2 Bestimmung relevanter Anwendungstypen und Repräsentanten

Zur Bestimmung relevanter Anwendungstypen wurde für die ausgewählten Prozessbereiche *Kunde* und *Vermittler* untersucht, welche Zugangswege existieren und welche Arten von Anwendungen eingesetzt werden (z.B. *dialogbasiert, ausgabeorientiert, informativ*). Zudem wurde deren Komplexität betrachtet. Das vorrangige Ziel lag dabei in der Abdeckung eines möglichst großen funktionalen Spektrums und damit der **qualitativen Auswahl relevanter und signifikanter Anwendungstypen** sowie Repräsentanten für die detaillierte Analyse. Die Relevanz und Signifikanz wurden aus quantitativen Betrachtungen abgeleitet.

Zur Bestimmung der Relevanz **dialogbasierter Anwendungen im Kundenumfeld**, wurden im Rahmen einer studentischen Arbeit [117] Online-Anwendungen unterschiedlicher Banken- und Versicherungen untersucht und der Anteil dialogbasierter Anwendungen ermittelt. Hierbei wurde ein Anteil dialogbasierter Anwendungen über alle Bereiche von ca. 70% festgestellt. 2019 wurde eine Nacherhebung mit Fokus auf die Anwendungen im Allianz-Kundenportal vorgenommen, bei der die Werte in Summe bereits bei ca. 95%[55] lagen. Bei der Nachuntersuchung wurde zudem eine Kategorisierung der Anwendungskomplexität vorgenommen.

Zur Analyse der **dialogbasierten Anwendungen im Vermittlerumfeld** wurden Ende 2012 die im Vermittler-Arbeitsplatz enthaltenen dialogbasierten Anwendungen untersucht. Hierbei wurde nach Art der Anwendung unterschieden und deren Komplexität bestimmt. Auf eine Nacherhebung wurde 2019 verzichtet, da sich keine signifikanten Änderungen ergeben

[55] Die Angabe bezeigt sich auf die Summe aller untersuchten Anwendungen im Kundenportal. Diese Zahl wird in den folgenden Abschnitten differenzierter betrachtet.

hatten. Lediglich die absolute Zahl der Anwendungen stieg, führte aber zu keiner Verschiebung der Verhältnisse bei den untersuchten Kategorien.

Die **Bewertung der Komplexität der Anwendungen** erfolgte anhand folgender Kriterien, die als Faktoren zur Gesamteinschätzung beitrugen:

- **UI-Elemente**: Komplexität der verwendeten Ein-/Ausgabe-Controls/Widgets
- **Dialoge und Abläufe**: Anzahl Felder und Darstellungseinheiten, Navigationsform
- **Verhalten**: Abhängigkeiten von Feldern, Ein-/Ausblenden von Feldern bzw. Darstellungseinheiten abhängig von angegebenen Werten
- **Backend-Bezug:** Zusammenspiel mit Backend-Systemen während der Verarbeitung, z.B. Bezug von Auswahlwerten, Berechnung von Zwischenergebnissen
- **Kombination / Einbettung:** Kombinierbarkeit mit bzw. Einbindung von anderen Anwendungen, z.B. Risikofragen in Lebensversicherungs-Antrag

Abschließend fand eine Auswahl repräsentativer Anwendungen statt, die zur Analyse der Interaktionsmuster in Kapitel 5 herangezogen wurden.

Anwendungen im Online-Kundenportal

Die Grundlage für die Analyse des Kundenbereichs bildeten die Anwendungen im Kundenportal der Allianz Deutschland AG[31]. Bei der Erfassung wurde nach drei Bereichen unterschieden:

- Offener Bereich (*http://allianz.de*)
- Geschlossener Bereich (*http://meine.allianz.de*)
- Self-Service-Anwendungen (*https://www.allianz.de/service/*)

Die Anwendungen im offenen Bereich sind allen Interessenten frei zugänglich, die im geschlossenen Bereich sind Kunden vorbehaltenen und erst nach Login im passwortgeschützten Bereich erreichbar. Unter Self-Service-Anwendungen werden Kontaktaufnahme-Formulare (sog. *Anliegen*) zusammengefasst, die sowohl im öffentlichen als auch angemeldeten Bereich als Anwendung *Servicecenter* angeboten werden. Diese wurden aufgrund ihrer hohen Zahl zugunsten einer qualitativen Bewertung gesondert erfasst, um eine Verzerrung der weiteren Ergebnisse zu vermeiden.

Tabelle A.1 stellt die bei der Analyse identifizierten Anwendungen Stand Ende 2019 zusammen. Insgesamt wurden **63 Anwendungen** und **140 Anliegen** erfasst. In der Tabelle sind die Anwendungen in die o.g. Bereiche untergliedert dargestellt und thematisch gruppiert. Für Bereiche und Gruppen ist die Summe sowie *Komplexitätstendenz* enthaltener Anwendungen festgehalten (Tendenzpfeile). Für jede Anwendung ist vermerkt, ob es sich um eine dialogbasierte Anwendung handelt sowie deren Komplexität indiziert. Die Bewertung erfolgt auf einer Skala von 0-4, wobei 4 die höchste Komplexitätsstufe darstellt. Die Darstellung erfolgt in Form teilgefüllter Kreise, wobei ein leerer Kreis den Wert 0 und ein voller den Wert 4 Repräsentiert. Die Werte 1-3 werden durch teilgefüllte Kreise dargestellt.

[31] Neben dem Allianz-Portal wurden bei der quantitativen Analyse anfangs weitere Versicherungs- und Finanzdienstleister-Portale einbezogen (z.B. Deutsche Kredit Bank, Deutsche-Bank, Techniker Krankenkasse, Barmer Ersatzkasse; vgl. Anhang A.1). Die Anwendungen sowie die Verhältnisse waren hier sehr ähnlich. Daher wurde die weitere Analyse auf die Allianz-Anwendungen beschränkt.

Anwendungen Online-Kundenportal

Anwendungen für Endkunden im offenen und geschlossenen Bereich des Allianz Deutschland Portals (www.allianz.de bzw. meine.allianz.de)	dialogbasiert	andere	Gesamt	Komplexität
Allianz - Offener Bereich	39	3	42	
Auto, Haus, Recht	16		16	
Anhänger-Versicherung	●			
Baufinanzierung Berechnung	●			
Baufinanzierung Budget-Rechner	●			
Baufinanzierung Online-Anfrage	●			
Hausrat Schadenmeldung	●			
Hausrat-Haftpflicht-Kombi Rechner	●			
Hausratversicherung	●			
KFZ Schadenmeldung	●			
Kfz-Schutzbrief	●			
Kfz-Versicherung	●			
Motorradsicherung	●			
Privat-Haftpflicht-Rechner	●			
Rechtschutz-Versicherung	●			
Verkehrs-Rechtschutz-Versicherung	●			
Wohnmobil-Versicherung	●			
Wohnwagen-Versicherung	●			
Beratung	3	3	6	
Agentursuche	●			
Online-Beratung (Terminvereinbarung)	●			
Online-Beratung Starten mit ID		●		
Online-Chat		●		
Zum Info-Service anmelden	●			
Produktfinder - Absprung in Antragstrecke		●		
Gesundheit & Freizeit	9		9	
Krankenhaus-Zusatzversicherung (mini-Rechner)	●			
Pflege-Rentenversicherung	●			
Pflege-Zusatzversicherung	●			
Private Krankenversicherung	●			
Reise-Gepäckversicherung	●			
Reise-Krankenversicherung	●			
Reise-Rücktrittversicherung	●			
Tierkranken-Versicherung	●			
Zahn-Zusatzversicherung	●			
Vorsorge & Vermögen	11		11	
Baufinanzierung	●			
Berufsunfähigkeits-Versicherung	●			
FORMORE -> PORTAL Absprung	●			
Park-Depot	●			
Private Rentenversicherung	●			
Riester-Rente	●			
Risiko-Lebensversicherung	●			
Rührup-Rente	●			
Sterbegeldversicherung	●			
Unfallversicherung	●			
Betriebliche Altersvorsorge (mini-Rechner)	●			

(Fortsetzung)	dialogbasiert	andere	Gesamt	Komplexität
Meine Alianz - Geschlossener Bereich	14	7	21	
Kondendokumente	1	1	2	
Anzeige		●		
Hochladen	●			
Meine Daten	4		4	
Kontaktdaten ändern	●			
Zahlungsdaten (Bankverbindung, Einzug, Zahlungsperiode)	●			
Persönliche Daten ändern	●			
Email-Statt Brief (Papierverzicht)	●			
Meine Verträge	3	1	4	
Bescheinigungen anfordern	●			
Freischaltung / Deaktivierung	●			
Kündigen	●			
Vertragsdetails & Absprünge		●		
Postfach	1	2	3	
Ausgang + Verwalten		●		
Eingang + Verwalten		●		
Nachricht senden	●			
Schaden	1	1	2	
Schaden melden	●			
Schadenübersicht		●		
Übergreifend	2	1	3	
Live-Chat		●		
Registrieren zu Meine Allianz	●			
Vermittlersuche	●			
Vorteilsprogramm	2	1	3	
Aktionscodes	●			
Einlösen von Prämien	●			
Prämienstatus		●		
Kunden-Selfservices	140		140	
Service-Center Formulare	140		140	
Anliegen	■		140	
Gesamt	193	10	203	

Auswertungen

Prozentsatz Dialogbasierte Anwendungen		
offener Bereich		92,9%
geschlossener Bereich		66,7%
Komplexitätsverteilung dialogbasierter Anwendungen		
offener Bereich	3	8%
	19	50%
	14	37%
	2	5%
geschlossener Bereich	1	7%
	9	64%
	4	29%
	0	0%

Tabelle A.1. Anwendungen im Online-Kundenportal

Im öffentlich zugänglichen Bereich wurden **42 Anwendungen** identifiziert, von denen ca. **93% dialogbasierte Anwendungen** i.S. der Arbeit darstellen. Bei den dialogbasierten Anwendung handelt es sich meist um Antragstrecken und Online-Rechner mit Dateneingabe. Die Komplexität der Anwendungen ist **moderat (50%)** bis **anspruchsvoll (37%)**. Ca. **5% stellen komplexe** Anwendungen dar, meist aufgrund starker Abhängigkeiten zu Backend-Systemen bzw. der Einbindung weiterer Anwendungen, z.B. zur Risikoanalyse.

Im geschlossenen Bereich wurden **21 Anwendungen** identifiziert, von denen ca. **67% als dialogbasiert** eingestuft wurden. Naturgemäß existieren hier viele Anwendungen, die informativen Charakter haben (z.B. Vertrags-, Kommunikations-, Schadenübersicht). Die dialog-

basierten Anwendungen sind hinsichtlich der Komplexität **moderat (64%)** bis **anspruchsvoll (29%)**

Die Kunden-Selfservices stellen mit **140 Anwendungen** den Löwenanteil und sind zu **100% dialogbasiert**. Aus Sicht der Generierung sind diese dadurch sehr relevant. Für die Analyse sind sie weniger interessant, da sie eine **überwiegend moderate, selten anspruchsvolle** Komplexität besitzen. Sie weisen meist geringe Dynamik und einfache Dialogstruktur auf und erfordern wenig bis keine Interaktion mit Backend-Systemen währen der Bearbeitung. Sie besitzen jedoch häufig Bereiche, die als wiederkehrende Bausteine genutzt werden (z.B. Adresseingabe, Vertragsauswahl etc.).

Im offenen Bereich überwiegen Anwendungen zur Angebotserstellung, sog. **Antragstrecken**. Deren Komplexität variiert abhängig vom zugrunde gelegten Produkt. So ist z.B. ein Riester-Rechner aufgrund der vom Gesetzgeber vorgegebenen Rahmenbedingungen sehr viel einfacher tarifierbar als eine Risiko-Lebensversicherung, bei deren Tarifierung Lebensumstände des Antragstellers essentiell für die Preisgestaltung sind (z.B. Freizeitaktivitäten, gesundheitliche oder berufliche Risiken). Im geschlossenen Bereich finden sich vorwiegend Anwendungen zur Information über den Zustand der Vertragsbeziehung. Nichtdialogbasierte Anwendungen zeigen Übersichten zu Verträgen, dialogbasierte Anwendungen dienen der Änderung persönlicher Daten (z.B. Adresse, Bankverbindung), Anforderungen (z.B. Bescheinigungen, Freischaltungen) oder Meldungen (z.B. Schadenmeldung, Kündigung, Kommunikation/Nachrichten an Vertreter oder Allianz).

Anwendungen auf dem Vermittlerarbeitsplatz

Die Grundlage zur Analyse der Vermittler-Anwendungen bildet der Vermittler-Arbeitsplatz als Desktop-Anwendung (*AMIS* - Allianz Makler Informations-System), wie er Ende 2012 ausgeliefert wurde. Ergänzend wurden differentielle Untersuchungen der Online-Anwendungen für Vermittler (*AMIS.online*) durchgeführt, die hier jedoch nicht explizit aufgeführt werden. Eine Nacherhebung war in diesem Bereich nicht notwendig, das sich die Produktlandschaft nur geringfügig geändert hat, wodurch die Aussagen der Analyse nicht verändert wurden.

Bei dieser Untersuchung wurden die Bausteine erfasst, die als dialogbasierte Anwendungen in den Rahmen des Vermittlerarbeitsplatzes integriert werden. Tabelle A.2 zeigt die bei der Erfassung identifizierten Anwendungen. Die Anwendungen sind **fachlich gruppiert** (*Versicherung, Vorsorge, Vermögen, Broschüren*) und zudem aufgeteilt in solche für **Privat- bzw. Firmenkunden**, die als Kundengruppen von Vermittlern bedient werden.

Zu jedem Baustein ist vermerkt, ob es sich um eine Anwendung zur **Angebotserstellung (Antragstrecke)**, ein **Werkzeug zur Beratung** oder um **Informationsanwendungen** (z.B. Formular-, Broschüren- und Content-Bereitstellung) handelt. Zudem ist eine Bewertung der Anwendungskomplexität angegeben. Die Bewertung erfolgt auf einer Skala von 0-4, wobei 4 die höchste Komplexitätsstufe darstellt. Die Darstellung erfolgt in Form teilgefüllter Kreise, wobei ein leerer Kreis den Wert 0 und ein voller den Wert 4 Repräsentiert. Die Werte 1-3 werden durch teilgefüllte Kreise dargestellt.

Insgesamt wurden **79 Bausteine** erfasst, von denen **52 Antragstrecken** (66%), **16 Beratungswerkzeuge** (20%) und **11 Informationsanwendungen** (14%) repräsentieren. Während sich die Anwendungen im Versicherungsbereich für *Privat- und Firmenkunden* die Waage

Anwendungen Vermittlerarbeitsplatz

Anwendungen des Vermittlerarbeitsplatzes (Desktop) mit Typisierung und Komplexität	Angebot	Beratung/Werkz.	Information	Gesamt	Komplexität
Versicherung	35	5	3	*43*	
Firmen	16	4	2	*22*	
Auto (Kraft)	●				
Auto Flotte	●				
Auto Handel + Handwerk	●				
Bau	●				
Bauherren-Haftpflicht	●				
Betriebshaftpflicht	●				
Elektronik	●				
Elementarzonenermittlung		●			
Ertragsausfall		●			
Firmenprogramm (AFP)	●				
Gebäudewertermittlung		●			
Haus	●				
Immobilien	●				
Inhalt			●		
Landwirtschaftl. Inhalt			●		
Maschinen	●				
Öko-Haftungsversicherung	●				
Rechtsschutz	●				
Transportversicherung	●				
Unfall - Gruppen		●			
Verkehrshaftung	●				
Vermögenschaden-Haftp	●				
Privat	19	1	1	*21*	
Auto (Kraft)	●				
auto.online service			●		
Automobilclub Deutschland	●				
Elementarzonenermittlung		●			
EnkelPolice 55Plus	●				
Erweiterte Haushalt	●				
Freizeit (Transport)	●				
Glas	●				
Haftpflicht	●				
Haus-/Wohnungsschutzbrief	●				
Hausrat	●				
Immobilien	●				
Link auf Elvia-Portal	●				
Moped	●				
Privatschutz	●				
Rechtsschutz	●				
Schutzbrief 55Plus	●				
Tierkranken (TKV)	●				
Unfall - Einzel/Familie	●				
Unfall - Gruppen	●				

(Fortsetzung)	Angebot	Beratung/Werkz.	Information	Gesamt	Komplexität
Vorsorge	14	10		*24*	
Firmen	3	1		*4*	
Arbeitnehmerangebotsprogramm	●				
BetriebsRente		●			
Firmenangebotsprogramm	●				
Rückdeckungsversicherung	●				
Privat	11	9		*20*	
BasisRente		●			
EnkelPolice 55Plus	●				
Kranken	●				
Leben	●				
Leben Allianz PortfolioPolice	●				
Leben Allianz RiesterRente und Allianz BasisRente	●				
Leben DLV	●				
Leben Privat	●				
MetallRente - Arbeitsnehmerangebotsprogramm	●				
MetallRente - Firmenangebotsprogramm	●				
MetallRente.Riester	●				
Produktvergleich		●			
RiesterRente		●			
Schutzbrief 55Plus	●				
Steuervorteilsrechner (Verkaufsassi. Vorsorge)		●			
Verkaufsassistent Kranken		●			
Versorgungs- und Anlageplanung (Wiederanlage)		●			
Vorsorgeanalyse		●			
ZeitWertkonto		●			
Zubehör		●			
Vermögen	3	1		*4*	
Privat	3	1		*4*	
Baufinanzierung	●				
Bausparen	●				
Investmentfonds	●				
Zubehör		●			
Broschüren			8	*8*	
Firmen			4	*4*	
Broschüre AAP			●		
Broschüre Firmen			●		
Broschüre Gruppen-Unfall			●		
Broschüre Pensionskasse			●		
Privat			4	*4*	
Broschüre Altersvorsorge			●		
Broschüre Investmentfonds			●		
Broschüre Leben			●		
Broschüre Leben RiesterRente und BasisRente			●		
Gesamt	52	16	11	*79*	

Tabelle A.2. Anwendungen im Vermittler-Arbeitsplatz

halten, dominieren im Vorsorge und Vermögens-Bereich die Anwendungen für Privatkunden.

Hinsichtlich der Komplexität befinden sich die Anwendungen auf einem **überwiegend anspruchsvollen Niveau**. Insbesondere die Antragstrecken als häufigste Anwendungsart sind tendenziell komplex, sodass diese Anwendungen für die Analyse von erhöhtem Interesse sind.

Potentielle Varianten: Überschneidungen

Tabelle A.2 zeigt indirekt auch Indikatoren für potentielle Varianten innerhalb der Vermittleranwendungen. So finden sich Bausteine für ein Produkt sowohl im Firmen- als auch im Privatkundenbereich (z.B. *Auto (Kraft), Rechtschutz, Gebäude, Haftpflicht*). Diese sind bisher als Einzelbausteine umgesetzt, können jedoch potentiell vereinheitlicht und als inhaltliche Varianten integriert werden.

Wie in Abschnitt A.1.1 beschrieben, existieren Anwendungen sowohl in der betrachteten Desktop-Anwendung als auch im Online-Portal AMIS.online. Hierbei handelt es sich um technologische Überschneidungen. Die Anwendungen selbst sind inhaltlich gleich, jedoch variiert die Umsetzung und Präsentation (Desktop vs. Web).

Auch mit den Kundenanwendungen aus Abbildung A.1 gibt es Überschneidungen. Hier existieren gemeinsame Anwendungen insbesondere für Antragstrecken im öffentlichen Bereich und für Beratungswerkzeuge, die als Varianten umgesetzt werden können (vgl. Beispiel für Vermittler- und Kundenanwendungen in Abschnitt A.1.1). Dabei handelt es sich sowohl um technologische als auch inhaltliche Varianten der Anwendungen.

Ergebnis:

Aufgrund obiger Betrachtungen wurden folgende Anwendungsarten als besonders geeignet für die automatische Generierung identifiziert und für die Durchführung der Analyse herangezogen:

- **Formularanwendungen** für Self-Services
- **Antragstrecken** zur Angebotserstellung für Versicherungs- und Vorsorgeprodukte
- **Erfassungsdialoge zur Risikoprüfung**

Diese Auswahl bildet einen breiten Querschnitt hinsichtlich der Komplexität der Abläufe, Struktur, UI-Elemente sowie der Dynamik der Oberflächen und ist damit eine solide Ausgangsbasis für die Analyse.

Formularanwendungen für Self-Services weisen einen einfachen Formularcharakter auf und haben die größte Verbreitung in der Anwendungslandschaft. Beispiele sind die Änderung einer Adresse oder Bankverbindung, Anfragen zu bestimmten Produkten/Dienstleistungen oder die Meldung eines Schadens.

Die Komplexität der Benutzungsschnittstelle ist gering. Meist handelt es sich um die Eingabe von Werten, die nur rudimentärer Prüfung bedürfen (z.B. syntaktischer Validierung, feste Wertebereiche für Eingaben). Varianten dieser Anwendungen beschränken sich meist auf die Bereitstellung für unterschiedliche Plattformen. Inhaltlich unterscheiden sich die Varianten nur gering für unterschiedliche Nutzergruppen.

Diese Anwendungen liefern Anforderungen an benötigte Benutzungsschnittstellen-Elemente und deren technische Varianten. Darüber hinaus kann die Wiederverwendung von Komponenten und die Kombination zu neuen Anwendungen untersucht werden.

Antragstrecken zur Angebotserstellung treten ebenfalls häufig auf. Sie dienen dem Neuabschluss eines Produktes und sammeln im Dialog mit dem Nutzer/Vermittler alle zum Antrag notwendigen Informationen. Die Dialoge sind hinsichtlich der Komplexität häufig an-

spruchsvoll und treten in Varianten auf, die sich sowohl inhaltlich als auch technisch unterscheiden. Sie werden beispielsweise vom Vertreter in einem Kundengespräch auf seinem Laptop als auch von Endkunden im Kundenportal des Versicherungsanbieters verwendet.

Im Vergleich zu den Self-Services ist die Komplexität der Benutzungsschnittstellen signifikant höher, da hier häufig eine Produkt-Konfiguration erfolgt, die abhängig von bereits eingegebenen Informationen weitere Nachfragen nach sich ziehen, wofür Programmlogik in der Oberfläche benötigt wird. Die Varianten für Antragstrecken unterscheiden sich stark hinsichtlich der Granularität der Fragen (nutzerspezifische Varianten). Sie werden meist in unterschiedlichen technischen Kontexten angeboten (z.B. Vertreter-Varianten für Laptop bzw. im Intranet; Endkunden-Varianten im Unternehmensportal oder für mobile Geräte).

Diese Anwendungen liefern Erkenntnisse über das dynamische Verhalten von Benutzungsschnittstellen und diesbezügliche Anforderungen an die Modellierung. Während die o.g. Self-Services grundlegende Eigenschaften liefern und einfach genug sind um Erkenntnisse zu Komponenten und Kompositionen zu liefern, vertiefen Antragstrecken Erkenntnisse über einzelne, komplexere Anwendungen.

Erfassungsdialoge zur Risikoprüfung sind im strengen Sinne keine eigene Anwendungskategorie, sondern Teil von Antragstrecken. Risikoprüfungen werden verwendet, um detailliert Risikofaktoren zu einem Versicherungsnehmer abzufragen (z.B. Gesundheitsfragen, ausgeübte Sportarten, Auslandsaufenthalte), die wiederum Einfluss auf den Produktpreis z.B. für eine Risikolebensversicherung haben. Zum Zeitpunkt der Analyse wurden diese noch in wenigen Endkunden-Anwendungen eingesetzt, fanden sich aber häufiger in Vermittler-Anwendungen. Im Zuge der Digitalisierung und Automatisierung werden Risikodialoge im Kundenumfeld jedoch immer relevanter, da eine automatische Risikoprüfung vermehrt in Antragstrecken für Kunden eingebaut werden müssen. Sie waren bei der Bewertung meist der **Grund für die Einstufung einer Anwendung als *komplex*** und wurden daher bei der Analyse näher untersucht.

Risikoprüfungs-Dialoge besitzen eine hohe Komplexität und Dynamik. Sie verwenden z.T. umfangreiche Nachfragen, die auf bereits erfolgten Eingaben und Zwischenberechnungen basieren. Die Komplexität der Benutzungsschnittstelle ist entsprechend hoch, da vermehrt Logik zur Steuerung der Oberfläche benötigt wird. Nutzergruppenspezifische Varianten dieser Anwendungsart unterscheiden sich ebenfalls in hohem Maße. So können Varianten für Vertreter und Makler detaillierter ausfallen, als Varianten für Endkunden im Unternehmensportal. Würde hier dieselbe Detailtiefe verwendet, ist es wahrscheinlich, dass der Kunde den Prozess vorzeitig abbricht.

Diese Dialoge liefern Erkenntnisse zu dynamischen Anwendungen, welche Abhängigkeiten zu Backend-Systemen haben und mit diesen kommunizieren müssen (z.B. zur Ermittlung von reaktionsrelevanten Zwischenergebnissen). Während die o.g. Antragstrecken Erkenntnisse für eine alleinstehende Anwendung liefern, können die komplexen Fragedialoge den Blickwinkel um dynamisches Verhalten, das von Drittsystemen abhängt, erweitern. Zudem liefern sie Erkenntnisse zur Integration von Dialogen in andere Anwendungen und deren Kombination

A.1.3 Auswahl betrachteter Technologien / Plattformen

Die bei Analyse betrachteten Technologien beschränkten sich auf vorhandene Umsetzungen der betrachteten Anwendungen der Allianz. Als technologische Varianten wurden daher folgende Kategorien betrachtet:

- **Desktop-Anwendungen** (Rich-Client und Browser-basiert)
- **Web-Anwendungen** (Intranet/Extranet und Portal-Bausteine)
- **Mobile, webbasierte Anwendungen**

Desktop-Anwendungen werden vorwiegend im Vertreterumfeld verwendet. Sie werden auf lokalen Umgebungen wie Desktop-Rechnern oder Laptops ausgeführt. Hierunter fallen *Rich-Client Anwendungen*, die mit nativen Technologien der Plattform erstellt werden (z.B. Java-Anwendungen mit JavaFX-basierten Oberflächen). Desktop-Anwendungen zeichnen sich durch eine vergleichsweise hohe Komplexität in der Oberfläche und Interaktion aus, da hier geringe Restriktionen bestehen. Beispielsweise bestehen vergleichsweise geringe Platzbeschränkungen, und es sind komfortable Eingabegeräte verfügbar. Zudem existieren Desktop-Anwendungen als *Browser-basierte Varianten*, die als Rahmenanwendungen in ihrer Oberfläche anspruchsvoll, aber mit Web-Technologien umgesetzt sind (z.B. AMIS und AMIS.online als Rahmenanwendungen).

Web-Anwendungen eigenen sich für unterschiedlich komplexe Anwendungen und können daher für alle Nutzergruppen sinnvoll eingesetzt werden. Die Ablaufumgebung ist hier ein (standardisierter) Browser und die verwendeten Technologien sind auf verschiedenen Geräten lauffähig. Web-Anwendungen besitzen eine hohe Relevanz für Unternehmen. Insbesondere der hohe Verbreitungsgrad der Technologie und die einfache Erreichbarkeit unterschiedlicher Nutzergruppen tragen zur Attraktivität dieser Anwendungsart bei. Die Ablaufumgebung für Web-Anwendungen (z.B. standardisierte Browser) ist heutzutage bei allen Nutzergruppen und auf den meisten Endgeräten verfügbar. Durch den hohen Verbreitungsgrad können Anwendungen mit dieser Technologie für alle Nutzergruppen erstellt und ohne Installationsaufwand verfügbar gemacht werden. Da Web-Abwendungen für alle Nutzergruppen relevant sind, variiert die Komplexität der Anwendungen im Vergleich zu den anderen Technologien am stärksten. Sie reicht von einfachen Formularen für Endkunden im Internet bis hin zu komplexen Browser-basierten Anwendungen für Vertreter oder Innendienst-Mitarbeiter, die als eigenständige Anwendungen z.B. im Intranet ausgeführt werden (z.B. AMIS.online-Bausteine)

Mobile, webbasierte Anwendungen sind Repräsentanten für eine Ablaufumgebung mit zahlreichen Restriktionen. Die Benutzungsschnittstellen sind starken Enschränkungen unterworfen, sodass deren Komplexität geringer ausfallen muss als z.B. bei regulären Webanwendungen. So existieren räumliche Restriktionen und die Eingabemöglichkeiten beschränken sich auf wenige Mittel oder nutzen alternative Eingabeformen (Touch-Bedienung, Bildschirmtastaturen, Spracheingabe). Daher werden sie meist für Anwendungen bzw. Varianten mit geringer Komplexität verwendet. Im Bereich der mobilen Anwendungen wurden webbasierte Varianten im mobilen Kundenportal der Allianz Deutschland untersucht.

Auf eine systematische Betrachtung nativer mobiler Anwendungen und sprachbasierter Anwendungen wurde in der Arbeit verzichtet, da zum Zeitpunkt der Untersuchungen keine repräsentativen Anwendungen der Allianz zum Vergleich zur Verfügung standen. Dennoch wurden - soweit möglich - informelle Abschätzungen für diese Technologien durchgeführt,

um diese nicht völlig außer Acht zu lassen. Daraus wurden folgende Eigenschaften abgeleitet und angenommen:

Native Mobile Anwendungen besitzen technologisch gesehen die Eigenschaften der o.g. Desktop-Anwendungen, da sie auf nativen Technologien beruhen, mit denen Anwendungen umgesetzt werden (z.B. iOS und Android). Jedoch gelten hier die o.g. Regeln und Restriktionen für mobile Anwendungen.

Sprachbasierte (dialogbasierte) Anwendungen können mit deklarativen Sprachen und Technologien erstellt werden wie dies im Verlauf der Arbeit mit o.g. Technologien erfolgte. Diese sind ebenfalls als native Technologien einzustufen. Die Restriktionen hinsichtlich der Komplexität sind hier noch stärker als bei mobilen Anwendungen und verbunden mit dem Bedarf nach klaren Navigations- und Orientierungsstrukturen. Es handelt sich danach um Anwendungen, die inhaltlich mobilen Anwendungen entsprechen, die technologisch weitergehenden Anforderungen entsprechen - jedoch grundsätzlich ins Muster dialogbasierter Anwendungen passen (Details hierzu siehe Kapitel 8, insbesondere Abschnitt 8.4.4).

Sofern sinnvoll wurden die informellen Ergebnisse in der Diskussion der Umsetzungskapitel der Arbeit angeführt.

A.2 mimesis Metamodelle

A.2.1 mimesis Metamodell für Anwendungen

Die folgenden Abbildungen und Tabellen beschreiben die Inhalte des Metamodells im Detail, das im Rahmen der Arbeit zur Beschreibung von Anwendungsmodellen vorgestellt wurde (vgl. Kapitel 6 und 7).

Abbildung A.6 führt die zur Modellierung vorgeschlagenen Elemente/Konzepte in Form eines UML-Diagramms zusammen. Die Darstellung gruppiert die Elemente in die Bereiche, zu deren Problemlösung sie beitragen (Modellierung der *Struktur*, der typisierten *Ein-/Ausgabe* und des *Verhaltens* der Benutzungsschnittstelle). Die Attribute der Elemente werden in den darauffolgenden Tabellen beschrieben.

In Tabelle A.3 sind die Attribute beschrieben, die in Abbildung A.6 für die strukturell relevanten Konzepte *DataDescription, DescriptionElement, DataGroup, DataComponent* und *DataItem* angegeben sind.

In Tabelle A.4 werden die Attribute zur Modellierung des Verhaltens (*Actions, Reactions, Validation, ModelCondition*) beschrieben.

Die Tabellen A.5, A.6 und A.7 erläutern abschließend die zur Modellierung der typisierten Ein-/Ausgabe vorgesehenen *Datentypen* und *Restriktionen* sowie die Attribute zur Spezifikation von *Semantic Tags*.

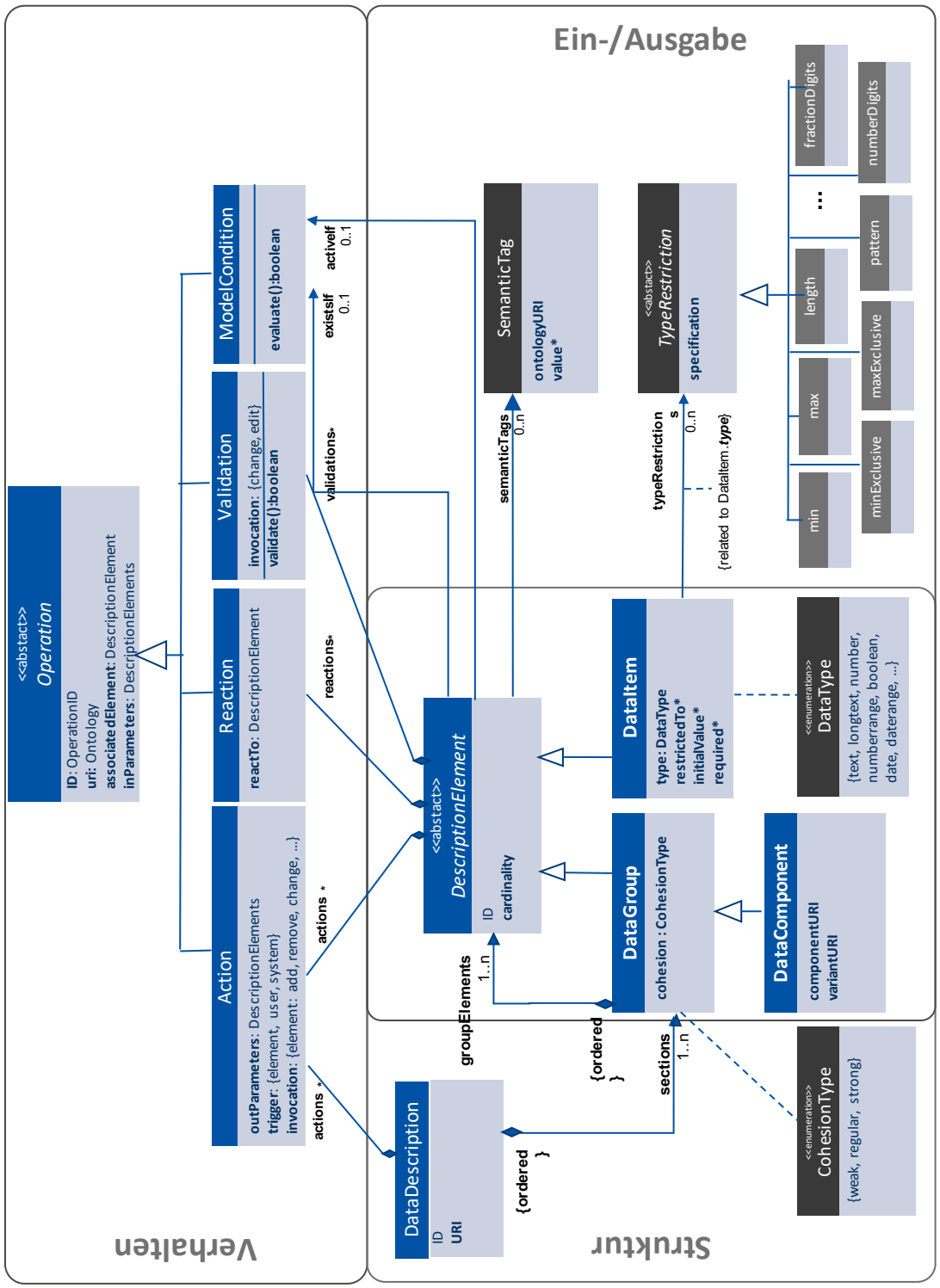


Abbildung A.6. mimesis Metamodell (Gesamtsicht)

Attribute	*Beschreibung*	*Ausprägungen*	*Notation in DSL*
DataDescription			
ID	Name des Anwendungsmodells	Zeichenkette [a-zA-Z0-9]+	
URI	eindeutige URI des Anwendungsmodells	URI	`uri="http://mimesis.soultions/aviation/flightbooking"`
DescriptionElement <abstract>			
ID	eindeutiger Name des Elements innerhalb seiner Hierarchiestufe. Dient der Identifikation des Elements innerhalb des Modells.	[a-zA-Z0-9]+	`"customerdata"` `"firstname"`
cardinality	Kardinalität der Gruppe oder des Elementwertes.	*siehe DataGroup / DataItem*	

existsIf	Bedingung für die Existenz des Elements. Evaluiert der Wert zu "wahr", sind die enthaltenen Daten relevant für die Bearbeitung und müssen erfragt werden.	**bool'scher Ausdruck** über Modellelementen oder **ModelCondition** als Operation im Anwendungskontext.	`existsIf=`
			`"(customer.ismarried==true) &&`
			`(customer.age<=60)"`
			`existsIf=`
			`"space.isApplicableTo (customer)"`

activeIf	Bedindung für die Veränderbarkeit des Elements. Evaluiert der Wert "wahr", sind die enthaltenen Daten editierbar. Sonst werden die enthaltenen Daten lediglich angezeigt.	**bool'scher Ausdruck** über Modellelementen oder **ModelCondition** als Operation im Anwendungskontext.	`active-if=`
			`"(customer.ismarried==true)"`
			`existsIf=`
			`"space.isMarried (customer)"`

semanticTags	Annotationen, die die Gruppe oder das Element semantisch beschreiben. Die verwendeten Annotationen sind in einer Ontologie beschrieben.	**SemanticTag**-Referenz.	`semanticTags="adress:zipcode"` `semanticTags="insure:bodyparts"`
DataGroup			
cohesion	Angabe der Kohäsion der enthaltenen Daten und der Gruppe zum Umfeld	**default:** keine Angabe. **strong:** die enthaltenen Daten haben eine starke Kohäsion und bilden eine Einheit (z.B. PLZ und Ort) **weak**: schwache Kohäsion zum Umfeld. Die enthaltenen Daten bilden eine vom Umfeld unabhängige Einheit und können bei Bedarf herausgelöst werden.	`cohesion="strong"` `cohesion="weak"`

cardinality	beschreibt die Kardinalität, mit der die Gruppe auftritt. Verwendung für Listen der beschriebenen Gruppe (z.B. die Angabe mehrer Adressen)	*: beliebige Anzahl an Vorkommen
		<*n*> : feste Anzahl
		<n>..<m>: Anzahl zwischen n und m.
			`cardinality="*"`
			`cardinality="3"`
			`cardinality="3..5"`

DataComponent			
componentURI	URI des Anwendungsmodells, das als Komponente eingebunden wird	URI	`uri="http://aviation.mimesis.solutions/products/flightbooking/v1"`
variantURI	URI des Variantenmodells und der konkret zu verwendenden Variante	URI	`variant="http://aviation.mimesis.solutions/products/flightbooking/v1/variants#travelagent"`
DataItem			

type	Typ des Datums
	Einfache Datentypen: Semantik gemäß XMLSchema simple data types
	Applikationsspezifische Datentypen: ggf. domänen- oder kontextspezifische Typen werden über ***semanticTags*** beschrieben, die auf einem einfachen Datentyp beruhen.
		einfacher Datentyp:
		text, number, boolean, longtext, date, float
			`type="number"`
			`type="text"`
			`type="date"`
			`type="boolean"`

+ Restriktionen	typbezogene Einschränkungen für Werte, Bereiche, Formate etc. (z.B. min/max, patterns)	s. Tabelle "Restriktionen"	`minInclusive="3"` `maxInclusive="10"`

restrictedTo	Einschränkung des Wertebereichs, der zur Auswahl steht.	Wertebereich abhängig vom angegebenen Typ.
		Optional kann der Bereich durch Aufruf einer Operation im Anwendungskontext befüllt werden.
			`restrictedTo="male|female"`
			`restrictedTo="0..10" (number)`
			`restrictedTo=`
			`"adresses.getZIP(address.city)"`

+ cardinality	gestattet die Auswahl mehrerer Werte aus einer mit restrictedTo spezifizierten Einschränkung (M-aus-N-Auswahl)	*: beliebige Anzahl an Vorkommen <*n*> : feste Anzahl <n>..<m>: Anzahl zwischen n und m.	`restrictedTo="one|two|three"` `cardinality="*"`
required	zeigt an, dass der Wert angegeben werden muss	true/false	`required="true"`

initialValue	Vorbelegung für den Wert des Elements	abhängig vom angegebenen Typ, ggf. vorhandener Einschränkungen und Multiplizität	`initialValue="male"`
			`initialValue="one|three"`
			`initialValue="2018-12-2"`

Tabelle A.3. Attribute der grundlegenden Modellelemente

Attribute	*Beschreibung*	*Ausprägungen*
Operation <abstract>		
ID	eindeutiger Name der auszuführenden Aktion	[a-zA-Z0-9]+
URI	URI der Operation in der Operationsontologie. Beschreibt die Semantik der Operation.	URI
associatedElement	Zuordnung zu Gruppe oder Element	**DescriptionElement-**Referenz
inParameters	Liste von Modellelementen, die als Parameter für die Aktion benötigt werden.	**DescriptionElement-**Referenzen
Action		
trigger	Auslöser der Aktion	**element**: Aktion, die durch Ereignisse am Element ausgelöst wird (s. invocation) **user:** Aktion, die durch Nutzer ausgelöst wird. **system:** Aktion, die durch die Laufzeitumgebung ausgelöst wird.
actionType	(optional) Typangabe für **element**-Aktionen. Beschreibt die Nutzung der Aktion näher (z.B. "info" für Hilfsfunktionen "preselect" für erweiterte Vorauswahl oder "pre-/postfix" für explizite Operationsoptionen vor/nach der Wertangabe)	**keine Angabe**: Standardverhalten **info**: Operation dient der Bereitstellung weiterer Informationen Hier ist eine detaillierte Spezifikation in zukünftigen Arbeiten erforderlich.
invocation	Auslösendes Ereignis für die Aktion	**element:** {change,enter,leave,edit, ...} **user:** {taptriggered,...} **system**: {dataloaded, ...}
outParameters	Liste von Modellelementen, die als Parameter von der Aktion verändert werden.	**DescriptionElement-**Referenzen
activeIf	Bedingung zur Ausführbarkeit der Aktion	**bool'scher Ausdruck** über Modellelementen oder **ModelCondition** als Operation im Anwendungskontext.
Validation		
invocation	Auslösendes Eregins für die Validierung	**edit**: während der Änderung **change:** bei Abschluss der Eingabe
Reaction		
reactTo	DescriptionElement, das beobachtet und auf dessen Änderung reagiert wird	**DescriptionElement-**Referenz
outParameters	Liste von Modellelementen, die als Parameter von der Aktion verändert werden.	**DescriptionElement-**Referenzen
ModelCondition		
keine Attributerweiterung notwendig		

Tabelle A.4. Attribute zur Spezifikation von Operationen

Attribute	*Beschreibung*	*Ausprägungen*	*Notation in DSL*
SemanticTag			

ontologyURI	URI der Ontologie, in welcher das angegebene Tag spezifiziert wird	URI	`referencedOntologies= [`
			`banking:`
			`  http://mimesis.solutions/types/ext/v1,`
			`life:`
			`  http://my.insurance/life/types/v1`
			`]`

value	Bezeichner des in der Ontologie referenzierten Elements	Elementname (Zeichenkette)	`banking:iban` `life:bodyparts`

Tabelle A.5. Attribute zur Spezifikation von SemanticTags

Datentyp	Verwendung	XMLSchema*	Beispiel
text	Textueller Wert - z.B. Vorname, Nachname, Stadt	#string	`Max Mustermann`

longtext	Komplexer textueller Wert. Der enthaltene Text kann mehrzeilig sein und ggf. Elemente einer Auszeichnungssprache beinhalten (z.B. Formatierung über HTML)	#string	`Dies ist ein Langtext.`
			
			`Dieser kann sich über mehrere Zeilen erstrecken. Der Inhalt kann <b>formatiert</b> sein.`

number
- integer
- decimal
	Numerischer Wert. Die Form wird durch Restiktionen näher bestimmt, z.B. ob es sich um einen ganzzahligen oder rationalen Zahlenwert handelt. Zur Vereinfachung kann der Typ "integer" bzw. "decimal" angegeben werden, welche entsprechende Restriktionen implizieren.	#integer
		#decimal
		#float
		#double
			`2007`
			`10.2`
			`121.42883`
			`2E12`

numberrange	Eingabe eines Wertebereichs mit unterem und oberem Wert. Analog dem Datentyp number kann auch hier der Typ "integerrange" bzw. "decimalrange" angegeben werden.	---	`(200,410)` `(123.34,2343.43)`
boolean	Angabe eines Wahrheitswertes (Ja/Nein-Feld)	#boolean	`true` `false`

date	Angabe eines Datums im ISO-Format [ISO 8601]	#date
		#dateTime
			`2017-10-10+13:00`
			`2017-10-10`
			`2017-01-15T12:00:00`

day, month, year, yearMonth, monthDay	Fragmente eines Datums	#gDay, #gMonth, #gYear, #gYearMonth, #gMonthDay	`2017` `2017-12`
time	Angabe eines Zeitpunktes (Uhrzeit)	#time	`12:00:00`
duration	Angabe einer zeitlichen Dauer	#duration	`02:30:00`
daterange	Angabe eines Datumsbereichs	---	`(2017-10-10;2017-10-11)`
timerange	Angabe eines Zeitabschnitts	---	`(12:00:00;14:00:00)`
*** Dokumentation: https://www.w3.org/TR/xmlschema-2/# ...**			

Tabelle A.6. Typen für Datenelemente

Restriktion	Beschreibung	Anwendbarkeit	Beispiel
Wertrestriktionen			
restrictedTo	Einschränkung auf eine feste Wertemenge des angegebenen Typs	alle*	{"Hund"|"Katze"|"Maus"} {"100€"|"250€"|"500€"}
restrictionCardinality	Multiplizität der Auswahl. Standard: 1	alle*	{0}, {1}, {*}, {0..n}, {1..7}
Bereichsrestriktionen			
min	minimaler Wert (#wert>=min)	number, date, time	30, 2017-08-08, 00:00:15
max	maximaler Wert (#wert<=max)	number, date, time	30, 2017-08-08, 00:00:15
minExclusive	minimaler Wert (#wert>minExclusive)	number, date, time	30, 2017-08-08, 00:00:15
maxExclusive	maximaler Wert (#wert<manExclusive)	number, date, time	30, 2017-08-08, 00:00:15
step	Schrittweite	number, date, time	1, 0.5, 00:15:00
unit	Mengeneinheit	alle*	"m","cm","kg","€"
Formatrestriktionen			
fractionDigits	Anzahl Nachkommastellen	number	2 (z.B. für Geldbeträge)
numberDigits	Anzahl Vorkommastellen	number	5 (z.B. für Postleitzahlen)
length, minLength, maxLe	Anzahl an Stellen/Zeichen	alle* - außer date, time und Derivate	300 (z.B. für Freitexteingabe)
pattern	Formatbeschränkung (Regulärer Asudruck)	alle* - außer date, time und Derivate	[a-Z1-9\.-]@[a-Z1-9\.-]\.[a-Z]{2,3} (z.B. für E-Mail)
Elementrestriktionen			
required	Indikator für ein Pflichtfeld		true
initialValue	initialer Wert des Feldes (abhängig vom Typ)	alle	12, "Hund", 1.2, 12:00:00, true, false
outputOnly	Ausgabefeld (keine Eingabe möglich)	alle	true, false
hidden	nicht sichtbares Feld	alle	true,false
* Ausgenommen: boolean			

Tabelle A.7. Restriktionen für Datenelementtypen

A.2.2 mimesis Metamodell für Varianten

Die folgenden Abbildungen und Tabellen beschreiben die Inhalte des Metamodells, das im Rahmen der Arbeit zur Beschreibung von inhaltlichen Varianten vorgestellt wurde (vgl. Kapitel 6.4). Abbildung A.7 zeigt das Variantenmodell beschränkt auf die untersuchten Modifikationen. In Tabelle A.8 werden die Attribute des Modells beschrieben.

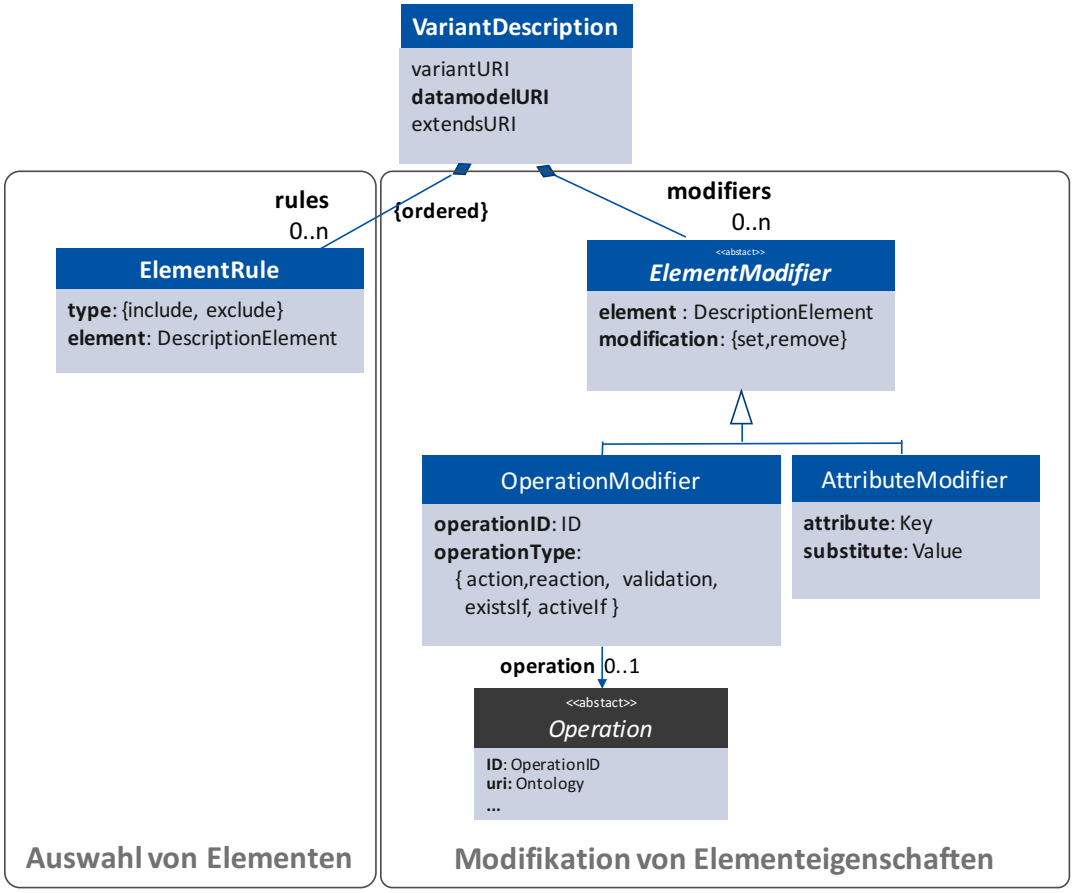


Abbildung A.7. mimesis Metamodell für Varianten (Gesamtsicht)

Attribut	*Beschreibung*	*Ausprägungen*	*Notation in DSL (Beispiele)*
VariantDescription			
variantURI	eindeutige URI des Variantenmodells für die spezifizierte Variante	URI	`uri="http://aviation.mimesis.soultions/flightbooking/v1/variants#travelagent"`
datamodelURI	eindeutige URI des Anwendungsmodells	URI	`uri="http://aviation.mimesis.soultions/flightbooking/v1"`
extendsURI	eindeutige URI des Variantenmodells, auf dem die Spezifikation aufsetzt (optional)	URI	`uri="http://aviation.mimesis.soultions/flightbooking/v1/variants#root"`
ElementRule			
element	Zu modifizierende Gruppe bzw. Element des Anwendungsmodells	**DescriptionElement**-Referenz	
type	Operation, die auf dem Modell auszuführen ist	{exclude, include} **exclude:** entfernt die angegebene Gruppe bzw. das Element aus dem Strukturbaum des Anwendungsmodells **include:** (re-)aktiviert die angegebene Gruppe bzw. das Element im Strukturbaum des Anwendungsmodells	
ElementModifier <abstract>			
element	Zu modifizierende Gruppe bzw. Element des Anwendungsmodells	**DescriptionElement**-Referenz	
modification	Modifikationsoperation, die auf der Gruppe bzw. dem Element auszuführen ist	{set,remove} **set:** modifiziert eine bestehende bzw. erzeugt eine neue Eigenschaft **remove:** entfernt die Eigenschaft	
AttributeModifier			
key	Name des zu modifizierenden Attributs	Alle Attribute, die für DataGroups bzw. DataItems im Metamodell angegeben sind.	
substitute	zu setzender Wert für das Attribut	Abhängig vom modifizierten Attribut (s. Metamodell, Datentypen, Einschränkungen)	
OperationModifier			
operationID	eindeutiger Name der zu modifizierenden Operation	(s. Metamodell)	
operationType	Typ/Kategorie der zu modifizierenden Operation im Anwendungsmodell	{action,reaction,validation,existsIf,activeIf}	
operation	Modellierung der modifizierten Operation gemäß Metamodell	abhängig von operationType: **Operation-, Action-, Reaction-, Validation-, ModelCondition**-Spezifikation (s. Metamodell)	

Tabelle A.8. Attribute der Elemente des Variantenmodells

A.2.3 mimesis Metamodell für Benutzungsschnittstellen (AUI/CUI)

In Abbildung A.8 ist das Metamodell zur abstrakten Modellierung von Benutzungsschittstellen in Form eines UML-Diagramms dargestellt.

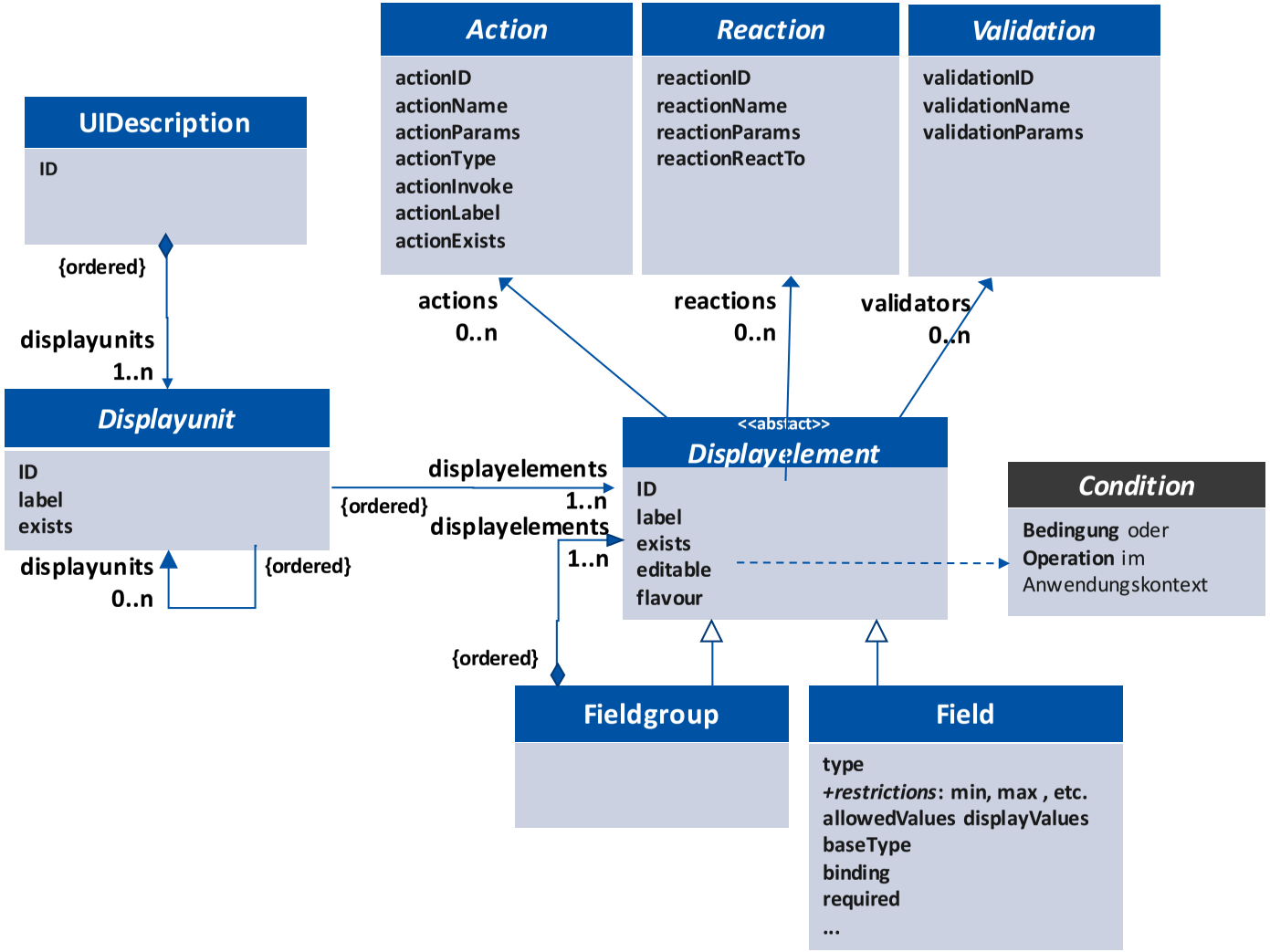


Abbildung A.8. mimesis Benutzungsschnittstellenmodell (Gesamtsicht)

Attribute	*Beschreibung*	*Ausprägungen*
UIDescription		
ID	Eindeutiger Bezeichner des UI-Models	Zeichenkette [.a-zA-Z0-9]+
_version	Version des UI-Modells	Zeichenkette
DisplayUnit		
ID	Eindeutiger Bezeichner der Darstellungseinheit	Zeichenkette [.a-zA-Z0-9]+
label	Textuelle Darstellung / Bezeichnung	Zeichenkette
exists	Bedingung für die Existenz der Darstellungseinheit. Evaluiert der Wert zu "wahr", ist die Darstellungseinheit sichtbar.	**bool'scher Ausdruck** über Modellelementen oder **Condition** als Operation des Anwendungskontexts.
DisplayElement <abstract>		
ID	Eindeutiger Bezeichner des Darstellungselements	Zeichenkette [.a-zA-Z0-9]+
label	Textuelle Darstellung / Bezeichnung	Zeichenkette
exists	Bedingung für die Existenz des Elements. Evaluiert der Wert zu "wahr", ist das Interaktionselement (Gruppe oder Datenelement) sichtbar.	**bool'scher Ausdruck** über Modellelementen oder **Condition** als Operation des Anwendungskontexts.
editable	Bedindung für die Veränderbarkeit des Interaktionselements. Evaluiert der Wert "wahr", sind die Daten editierbar. Sonst werden die enthaltenen Daten lediglich angezeigt.	**bool'scher Ausdruck** über Modellelementen oder **Condition** als Operation des Anwendungskontexts.
flavour	Darstellungsvariante des Interaktionselements für Gruppe oder Datenelement	Variantenbezeichnung des Interaktionselements (vordefinierte Menge bzw. als **SemanticTag**-Referenz)
FieldGroup		
keine zusätzlichen Attribute (s. DisplayElement)		
Field		
type	Typ des Interaktionselements.	text, number, boolean, longtext, date, oneOfMany, manyOfMany, ... (s. Tabelle "Interaktionselemente")
+ Restriktionen	typbezogene Einschränkungen für Werte, Bereiche, Formate etc. (z.B. min/max, patterns)	s. Tabelle "Restriktionen"
binding	Referenziertes Element des Datenmodells. Modelliert die Verbidnung des Interaktionselements zur Datenhaltung im Anwendungskontext.	Zeichenkette [.a-zA-Z0-9]+
allowedValues ***displayValues***	Einschränkung zur Auswahl stehender Werte sowie dazustellender Texte (labels) für diese Werte.	Wertebereich abhängig vom angegebenen Basis Typ (s. baseType) Optional können Werte und Texte durch Aufruf einer Operation im Anwendungskontext befüllt werden.
baseType	zugrundeliegender Basistyp bei oneOfMany bzw. manyOfMany Feldern	(s. Tabelle "Interaktionselemente")
value	Vorbelegung für den Wert des Elements	abhängig vom angegebenen Typ, ggf. vorhandener Einschränkungen und Multiplizität

Tabelle A.9. Attribute der Modellelemente des Benutzungsschnittstellenmodells

Interaktionselement	*Beschreibung*
Basistypen	
number	numerischer Wert
text	textueller Wert
longtext	Bereich für längere textuelle Werte
date	Datum
time	Zeitangabe
boolean	bool'scher Wert
oneOfMany	Auswahl eines Wertes aus einer Liste (1-aus-N-Auswahl)
manyOfMany	Auswahl mehrerer Werte aus einer Liste (M-aus-N-Auswahl)
actionElement	Explizites Element für Nutzeraktion (z.B. Schaltfläche)
Erweiterte Typen	
numberminmax	beschränkter numerischer Wert
numberrange	numerischer Wertebereich
daterange	Datumsbereich
timerange	Zeitspanne

Restriktion	*Beschreibung*
numerische Werte	
min / minExclusive	minimaler Wert
max / maxExclusive	maximaler Wert
step	Schrittweite
unit	Einheit des Werts
fractionDigits	Nachkommastellen
textuelle Werte	
length, min-/maxLength	Anzahl Zeichen bei textueller Eingabe
regex / regexText	regulärer Ausdruck zur Festlegung des Formats und Fehlermeldung bei Fehleingabe.
generell	
required	Eingabe erforderlich
displayonly	Nur Anzeige, keine Eingabe möglich
hidden	Element wird nicht angezeigt

Tabelle A.10. Attribute der Modellelemente (2)

Attribute	*Beschreibung*	*Ausprägungen*
Reaction		
reactionID	Eindeutiger Bezeichner der Operation	Zeichenkette [.a-zA-Z0-9]+
reactionName	Name der Operation im Anwendungskontext	**Referenz auf Operation** im Anwendungskontext
reactionParams	Liste von Modellelementen im Anwendungskontext, die als Parameter von der Aktion verwertet werden.	**Referenzen auf Datenelemente** im Anwendungskontext.
reactionReactTo	Modellelement, dessen Änderung die Reaktion auslöst	**Referenz auf Datenelement** im Anwendungskontext.
Action		
actionID	Eindeutiger Bezeichner der Operation	Zeichenkette [.a-zA-Z0-9]+
actionName	Name der Operation im Anwendungskontext	**Referenz auf Operation** im Anwendungskontext
actionParams	Liste von Modellelementen im Anwendungskontext, die als Parameter von der Aktion verwertet werden.	**Referenzen auf Datenelemente** im Anwendungskontext.
actionType	Typ der Operation, der deren Nutzung und damit Repräsentation in der Benutzungsschnittstelle bestimmt.	**element** : Standardrepräsentation **info**: Hilfe zum Element **pre-/postfix**: z.B. explizit dargestelltes Interaktionselement vor bzw. nach der Eingabe. +***zukünftige Erweiterungen*** (z.B. Funktionalität für erweiterte Vorauswahl)
actionInvoke	Auslösendes Eregnis für die Aktion	*select, change, enter, leave, edit* , etc.
actionLabel	Textuelle Darstellung / Bezeichnung in der Benutzungsschnittstelle (Label)	Zeichenkette
actionExists	Bedingung zur Anzeige der Aktion	**bool'scher Ausdruck** über Modellelementen oder **Condition** als Operation im Anwendungskontext.
Validation		
validationID	Eindeutiger Bezeichner der Operation	Zeichenkette [.a-zA-Z0-9]+
validationName	Name der Operation im Anwendungskontext	**Referenz auf Operation** im Anwendungskontext
validationParams	Liste von Modellelementen im Anwendungskontext, auf die lesend zur Validierung zugegriffen wird.	**Referenzen auf Datenelemente** im Anwendungskontext.

Tabelle A.11. Attribute der Modellelemente (3)

A.3 mimesis Beschreibungssprachen

Dieses Kapitel gibt einen Überblick über die Beschreibungssprachen, die zur Repräsentation der im Rahmen der Arbeit entwickelten Modelle genutzt werden. Es handelt sich hierbei um eine Einführung, die zum besseren Verständnis der Beispiele in der Arbeit betragen soll. Die folgenden Beschreibungen stellen keine umfassende Dokumentation der Beschreibungssprachen dar, sondern reflektieren den Stand der prototypischen Implementierung, die im Zuge der Arbeit erstellt wurde.

A.3.1 Anwendungsmodell als DSL (mimesis.model.DSL)

Das Anwendungsmodell dient zur Beschreibung der verarbeiteten Daten einer Anwendung, aus welchen schrittweise eine finale Benutzungsschnittstelle abgeleitet werden kann. Die Beschreibung des Anwendungsmodells folgt dem in Anhang A.2.1 zusammengefassten Metamodell und stellt eine textuelle Form der darin enthaltenen Informationen dar.

In Abbildung A.9 ist der Aufbau einer Anwendungsbeschreibung schematisch dargestellt. Er gliedert sich in zwei Bereiche: Metainformationen (*Metadaten*) zum Modell und die modellierten Knoten des Strukturbaums (*Sektionen*).

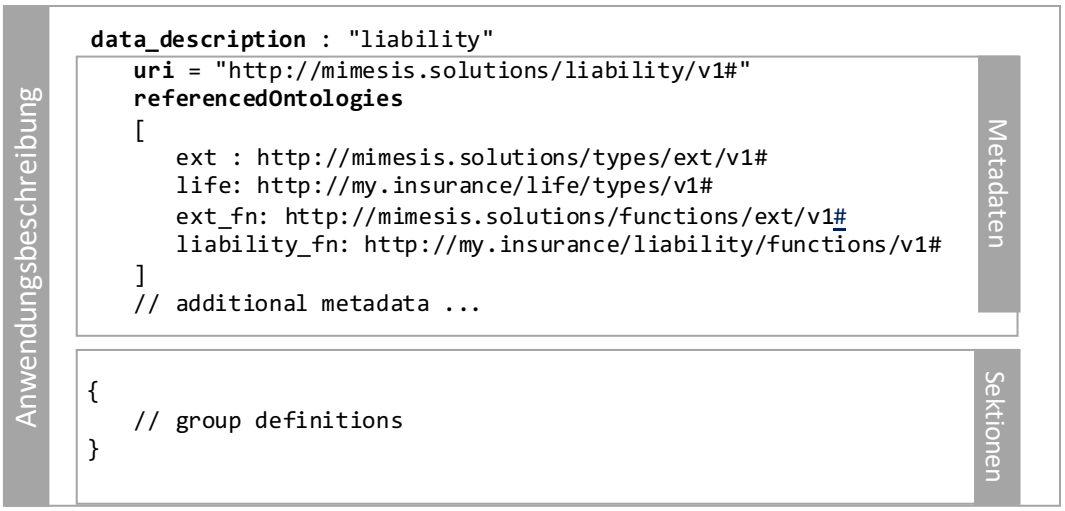


Abbildung A.9. Referenzierte Ontologien der Anwendungsbeschreibung

Der **Bereich Metadaten** enthält allgemeine Informationen zum Modell und dessen Nutzungskontext. Dies können z.B. Informationen zur Version, dem Ersteller des Modells oder zum Anwendungskontext/Domäne des Modells sein. Viele dieser Metainformationen sind organisatorischer Natur und werden nicht direkt zur Herleitung einer Benutzungsschnittstelle benötigt. An dieser Stelle beschränkt sich die Darstellung auf für die Herleitung relevanten Metainformationen. Abbildung A.9 zeigt die Nutzung des Metadaten-Bereichs am Beispiel der Informationen zu Ontologien, die zur Beschreibung der Semantik von Strukturbaumknoten sowie für Operationen aus dem Anwendungskontext referenziert werden.

In der Abbildung sind Ontologien aufgezählt, die zur semantischen Beschreibung von Gruppen und Datenelementen verwendet werden (*referencedOntologies*). Zu jeder Ontologie wird eine Kurzbezeichnung angegeben, mit welcher Bezug auf die Ontologie genommen wird. In den Metainformationen wird z.B die Ontologie *http://mimesis.solutions/types/ext/v1#* referenziert, die eine semantische Definition für *contactInfo* und *gender* beinhaltet. Beide kön-

nen als semanticTags an Modellelementen über *ext:contactInfo* und *ext:gender* genutzt werden.

Beschreibung der Struktur

Der **Bereich Sektionen** enthält eine textuelle Repräsentation des Strukturbaums und der darin enthaltenen Elemente. Die Gruppierung wird durch die Zusammenfassung semantisch zusammenhängender Elemente in *group*-Bereichen beschrieben. Diese können geschachtelt sein und bilden eine Hierarchie, in der die Elemente einer Hierarchiestufe die gleiche Kohäsion aufweisen. Abbildung A.10 stellt dies schematisch dar. Hier ist eine *Anwendungsbeschreibung* (*data_description*) als Folge von *Gruppen* (*group*) dargestellt.

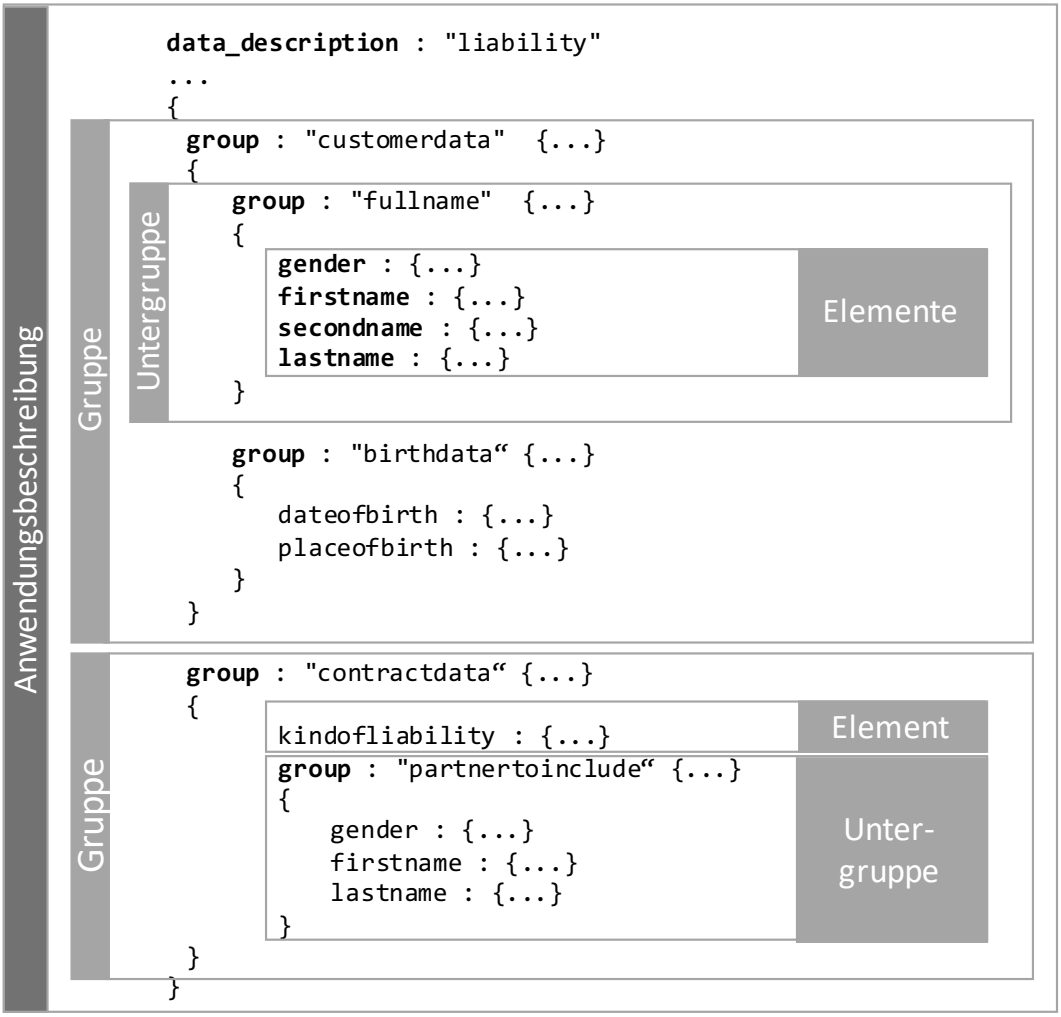


Abbildung A.10. Struktureller Aufbau der Anwendungsbeschreibung

Jede Gruppenbeschreibung wird durch ***group*** und einen Gruppenbezeichner (*ID* im Metamodell) eingeleitet. Diesem folgt ein Attributblock, der die Eigenschaften der Gruppe beschreibt (in Abbildung A.10 durch *{...}* angedeutet). Diesem folgt ein Block, der die Elemente der Gruppe enthält. Das können weitere Gruppen (*Untergruppen*) oder Datenelemente sein. Datenelemente werden über ihren Bezeichner (*ID* im Metamodell), gefolgt von einem Attributblock beschrieben. Die **Sequenz der Elemente** (*{ordered}*-Eigenschaft im Metamodell) ist über deren Reihenfolge innerhalb der Gruppe festgelegt.

Beschreibung der Eigenschaften von Gruppen

Die im Metamodell vorgesehenen Eigenschaften einer Gruppe werden im Attributblock angegeben. Abbildung A.11 zeigt den schematischen Aufbau einer Gruppenbeschreibung. Jede Gruppe wird über einen Gruppenbezeichner bestimmt, gefolgt von zwei Bereichen, die jeweils in geschweifte Klammern eingeschlossen sind. Der erste Block enthält die Attribute der Gruppe. Der zweite Bereich enthält die Elemente, die der Gruppe untergeordnet sind.

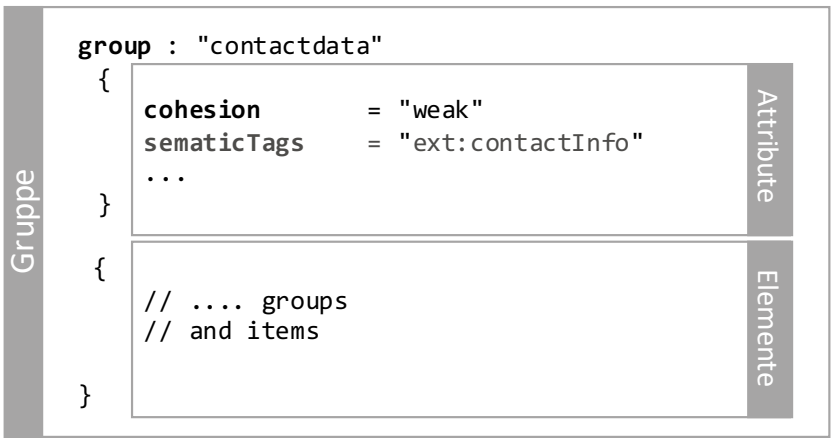
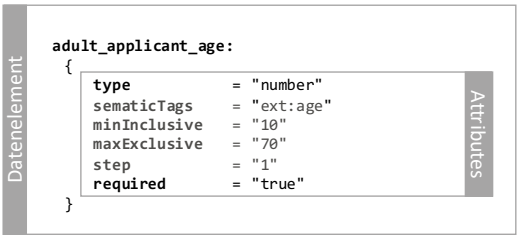


Abbildung A.11. Beschreibung von Gruppen

Die im Attributbereich spezifizierbaren Angaben entsprechen dabei den in Abschnitt A.2.1, Tabelle A.3 beschriebenen Attribute. Abbildung A.11 zeigt dies am Beispiel der Kohäsion (*cohesion*) und der Angabe eines *SemanticTag* für die Kontaktinformationen (*ext:contactInfo*).

Beschreibung der Eigenschaften von Datenelementen

Analog der Attributsbeschreibung bei Gruppen werden Attribute für Datenelemente in einem Attributbereich angegeben.



(a) Altersangabe mit Restriktionen

(b) Angabe erkrankter Körperbereiche

Abbildung A.12. Beschreibung von Datenelementen

Abbildung A.12 zeigt den schematischen Aufbau einer Beschreibung für Datenelemente am Beispiel einer Altersangabe mit Wertbeschränkungen und zur Angabe erkrankter Körperbereiche. Beiden Datenelementen werden Typinformationen und geltende Restriktionen zugewiesen, wie sie im Metamodell Abschnitt A.2.1, Abbildung A.6 dargestellt sind. Die Angabe erfolgt dabei textuell in Form von Attribut-Wert-Paaren. Auch die im Metamodell als assoziierte Objekte modellierten Einschränkungen werden als direkte Attribute angegeben (z.B. Abb. A.12 (a), *minInclusive*). Analog werden *semantic Tags* spezifiziert.

Beschreibung des Verhaltens

Die Beschreibung des Verhaltens erfolgt für Gruppen- und Datenelemente über die Angabe von Attributen, die in deren Attributblock angegeben werden. Hierfür sind analog den Relationen-Namen des Metamodells (Abbildung A.6) die Attribute *existsIf*, *activeIf*, *validations*, *actions* und *reactions* vorgesehen, die Listen von Operationsobjekten enthalten können. Für Existenz-/Sichtbarkeit, Validierungen, Aktionen und Reaktionen unterscheiden sich die Inhalte der Objekte gemäß dem Metamodell aus Abschnitt A.2.1, Abbildung A.6. Die möglichen Attribute für jeden Operationstyp sind im selben Abschnitt in Tabelle A.4 zusammengefasst.

Existenz

```
group: "partner_information" {
 existsIf= [
    {
       id      : "existsIf"
       uri     : "liability_fn:exists_partner"
       in      : "<path.to>.isMarried"
    }
 ]
}
```

Validierung

```
iban: {
 validations = [
    {
       id          : "ibanvalid"
       uri         : "ext_fn:ibanvalid"
       invocation  : "change"
       in          : "<path.to>.iban"
    } ... // additional validations
 ]
}
```

Aktion

```
userdata: {
 actions = [
    {  id          : "operUserDB"
       trigger     : "user"
       invocation  : "control"
       uri         : "agent_fn:openUserDB"
       in          : "<path.to>.lastname"
       out         : "<path.to>.firstname,
                      <path.to>.lastname, ... "
    } ... // additional actions
 ]
}
```

Reaktion

```
city : {
 reactions = [
    {  id          : "prefillCity"
       reactTo     : "<path.to>.zip"
       uri         : "ext_fn:prefillCity"
       in          : "<path.to>.zip"
       out         : "<path.to>.city"
    } ... // additional reactions
 ]
}
```

Abbildung A.13. Beschreibung des Verhaltens von Gruppen / Datenelementen

Abbildung A.13 stellt den aus dem Metamodell resultierenden Aufbau der Beschreibungen für die Verhaltensarten dar. Sie zeigt die Beschreibung der Operationsarten als Attribut-Wertpaare und deren Zusammenfassung in Objekten. Diese Objekte können als Listen zusammengefasst und den Verhaltensattributen zugewiesen werden. Die Beschreibung einer Validierung erfolgt hier z.B. anhand der Attribute *id*, *uri*, *invocation* und der Liste der *in*-Parameter der Operation. Diese werden in geschweifte Klammern eingefasst als Objekt beschrieben. Die Zuweisung an das Attribut *validations* erfolgt in einer Liste, welche kommasepariert und durch eckige Klammern eingeschlossen ist (*validations = [...]*).

Da diese Darstellung der Operationsbeschreibung ggf. zu komplexen Anwendungsbeschreibungen führt, wird im Rahmen der prototypischen Implementierung eine kompaktere Syntax zur Definition der Operationen verwendet. Diese fasst die Attribute einer Operation in einer Zeichenkette zusammen. In Abbildung A.14 ist die Syntax der vereinfachten Beschreibung dargestellt. Zudem werden die in Abbildung A.13 dargestellten Beispiele in der vereinfachten Notation dargestellt.

Existenz	**existsIf/activeIf="** *<boolean expressison of <in.list> modelements>* **"**
Validierung	**validations="** *<invocation>* **:** *<uri>* **(** *<in.list>* **); ..."**
Aktion	**actions="** *<trigger>* **:** *<invocation>* **:** *<uri>* **(** *<in.list>*, **$***<out.list>*);..."
Reaktion	**reactions = "***<reactTo>* **:** *<uri>* **(** *<in.list>*, **$***<out.list>* **); ..."**

Existenz

```
group: "partner_information" {
  existsIf =
     "(<path.to>.isMarried == true) && ... )"
}
```

Validierung

```
iban: {
  validations=
     "change:ext_fn.ibanvalid(<path.to>.iban);..."
}
```

Aktion

```
userdata: {
  actions = "user:control:agent_fn.open_userDB(
                <path.to>.lastname,
                $<path.to>.firstname,
                $<path.to>.lastname ); ..."
}
```

Reaktion

```
city : {
  reactions = "<path.to>.zip:ext_fn.prefillCity(
                <path.to>.zip,
                $<path.to>.city); ..."
}
```

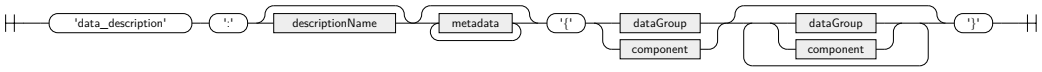
Abbildung A.14. Beschreibung des Verhaltens von Gruppen / Datenelementen (Kurzform)

Syntax mimesis.model.DSL

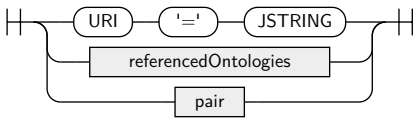
Die folgenden Syntax-Diagramme beschreiben die Syntax der *mimesis.model.DSL*, wie sie im Rahmen der Implementierung zur Evaluation umgesetzt wurde. Da zum Zeitpunkt der Entwicklung die konkrete Modellierung einiger Aspekte noch nicht final festgelegt war, finden sich z.T. noch generische Konstrukte in der Grammatik (z.B. die Verwendung einfacher Strings anstatt konkreter Festlegungen für Operationsdefinitionen oder möglicher Wertausprägungen). Die Syntaxbeschreibung ist daher in dieser Form nicht zur Definition eines Standards geeignet und wird in zukünftigen Versionen finalisiert.

model:

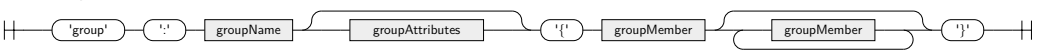
dataDescription:



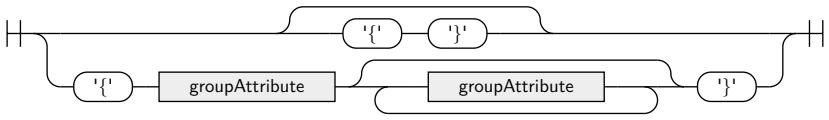
metadata:



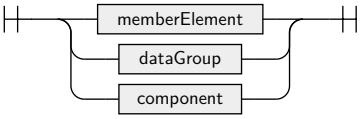
dataGroup:



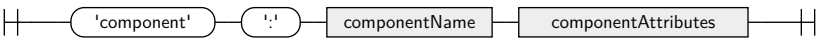
groupAttributes:



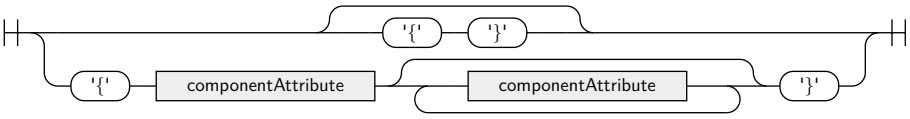
groupMember:



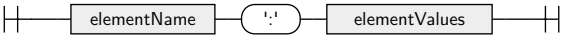
component:



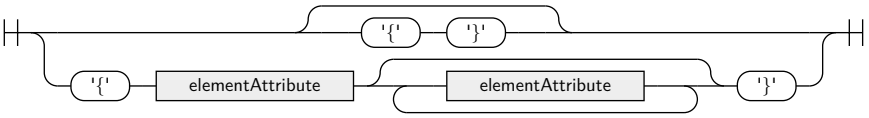
componentAttributes:



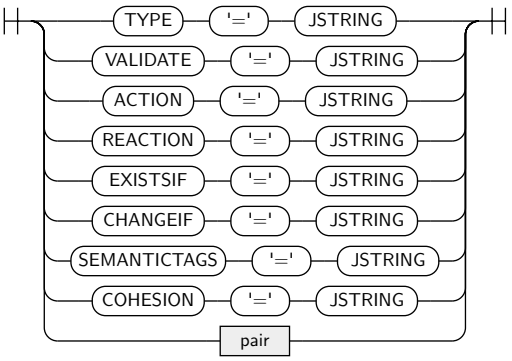
memberElement:



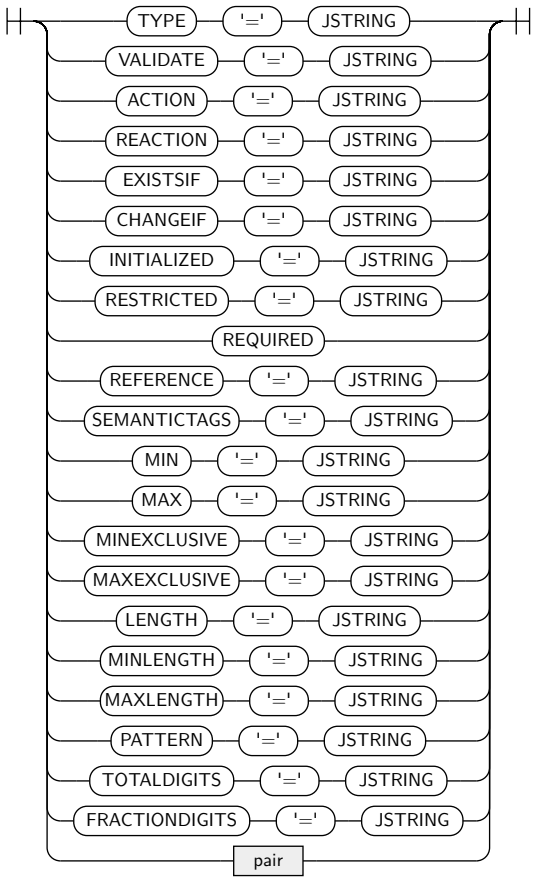
elementValues:



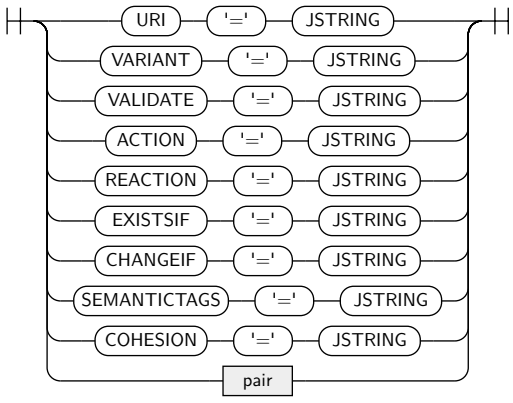
groupAttribute:



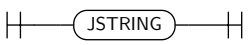
elementAttribute:



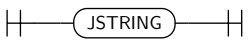
componentAttribute:



descriptionName:



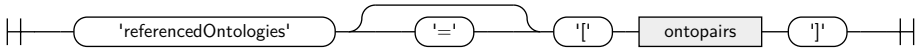
groupName:



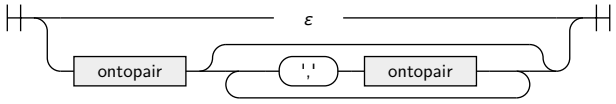
componentName:

elementName:

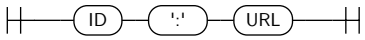
referencedOntologies:



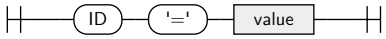
ontopairs:



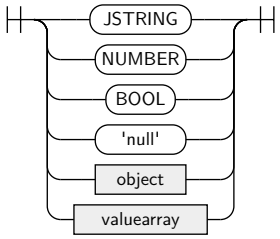
ontopair:



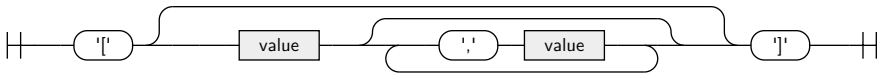
pair:



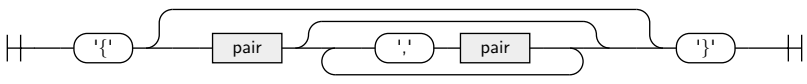
value:



valuearray:



object:



Terminale

URI	uri	**MIN**	min
TYPE	type	**MAX**	max
INITIALIZED	initialValue	**MINEXCLUSIVE**	minEclusive
RESTRICTED	restrictedTo	**MAXEXCLUSIVE**	maxExclusive
REQUIRED	required	**LENGTH**	length
EXISTSIF	existsIf	**MINLENGTH**	minLength
CHANGEIF	activeIf	**MAXLENGTH**	maxLength
ACTION	actions	**PATTERN**	pattern//checkifneeded
REACTION	reactions	**TOTALDIGITS**	totalDigits
VALIDATE	validations	**FRACTIONDIGITS**	fractionDigits
REFERENCE	reference	**VARIANT**	variant
SEMANTICTAGS	semanticTags		

A.3.2 Variantenbeschreibung als DSL (mimesis.variant.DSL)

Die Beschreibung von Varianten erfolgt in einer oder mehreren strukturierten Dateien, in denen mehrere Variantenspezifikationen zusammengefasst werden. Da Varianten ggf. von Dritten erstellt werden, erfolgt deren Spezifikation nicht im Anwendungsmodell selbst.

Die Variantenbeschreibung folgt dem in Anhang A.2.2 dargestellten Metamodell und stellt eine textuelle Form der dort enthaltenen Informationen dar. In Abbildung A.15 ist der Aufbau einer Variantenbeschreibung schematisch dargestellt.

Eine Variantenbeschreibung (*variant_description*) wird durch einen Metadaten-Block eingeleitet, der grundlegende Informationen zur Beschreibung enthält. Hier ist einerseits die URI des Variantenmodells (*uri*) andererseits die URI des Applikationsmodells angegeben, für welches Variantenspezifikationen enthalten sind (*datamodelURI*). Zudem sind analog der Anwendungsbeschreibung (vgl. Abschnitt A.3.1) Ontologien angegeben, die im Verlauf der Beschreibung referenziert werden. Stellt die Variante eine Erweiterung einer Bestehenden dar, ist zudem die URI der Basis-Variante angegeben (*extends*).

Abbildung A.15. Struktur des Variantenmodells

Auf den Metadaten-Bereich folgt eine Reihe von Variantenspezifikationen (*variant*), die über einen Namen und eine eindeutige *uri* identifiziert werden. Jede Variantenspezifikation gliedert sich in drei Bereiche (vgl. Abbildung A.16). Dies sind *Metadaten* zur Variante, *Regeln* zur Auswahl relevanter Elemente und *Modifikatoren* zur Anpassung von Eigenschaften und des Verhaltens der Variante (vgl. Abschnitt 6.4).

Abbildung A.16. Struktur der Variantenspezifikation

Elementregeln: Auswahl von Elementen

Die Elementregeln (vgl. Anhang A.2.2, *ElementRules*) beschreiben die Inklusion bzw. Exklusion von Gruppen oder Elementen des Anwendungsmodells in der Variante. Hierbei wird zuerst die Operation (*include* bzw. *exclude*) und anschließend der Pfad auf das betreffende Element im Anwendungsmodell angegeben. Die Regeln werden dabei in der angegebenen Reihenfolge angewendet. Die Beschreibung enthält damit alle Informationen, die im Metamodell für Elementregeln spezifiziert wurden (vgl. Tabelle A.8, *ElementRule*))

Modifikatoren: Anpassung von Elementeigenschaften und Verhalten

Die Anpassung von Elementeigenschaften und des Verhaltens erfolgt über Modifikatoren (vgl. Anhang A.2.2, *ElementModifier*). Diese werden als *modify*-Einträge gefolgt von einer in geschweifte Klammern eingefassten Beschreibung der Modifikation in der Variantenspezifikation beschrieben. Hierbei wird zwischen Attribut- und Operationsmodifikatoren unterschieden. Die *modify*-Einträge enthalten dabei die in Abbildung A.7 und Tabelle A.8 dargestellten Informationen. Abbildung A.17 zeigt exemplarisch den Aufbau der Beschreibungen.

Das Feld *type* bestimmt, ob es sich um eine Attribut (*attribute*) oder Operationsmodifikation (*operation*) handelt. Für alle Modifikationen wird gemäß dem Modell in den Feldern *element* und *modification* angegeben, für welches Element des Anwendungsmodells die Modifikation gilt und ob ein neuer Wert gesetzt oder ein bestehender entfernt werden soll (*set, remove*).

Abbildung A.17. Attributs- und Operationsmodifikation

Die Beschreibung der **Attributmodifikation** erfolgt unter Angabe des Namens des zu modifizierenden Attributs des Elements (*attribute*). Hier wird der Name des Gruppen- bzw. Elementattributs des Anwendungsmodells (vgl. Anhang A.2.1, Tabelle A.3) angegeben. Mit *newvalue* wird der neue Wert angegeben, den das Attribut zukünftig tragen soll.

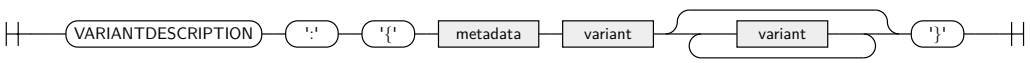
Zur Beschreibung einer **Operationsmodifikation** wird die Operationsart (*operationType*: *action, reaction, validation*) sowie die ID der auszutauschenden Operation innerhalb dieses Bereichs angegeben (vgl. Anhang A.3.1, Abbildung A.13). Für jede Operationsart die in der Variante zu verwendende Operation (*_operation*) sowie deren Ein- und Ausgabeparameter (*_parameters*) spezifiziert. Für eine **Aktion** wird darüber hinaus der Aktionstyp und das auslösende Ereignis (*action_type, action_invocation*) angegeben. Für eine **Reaktion** wird die Referenz auf das Element angegeben, auf dessen Zustand reagiert wird (*reaction_reactto*).

Syntax mimesis.variants.DSL

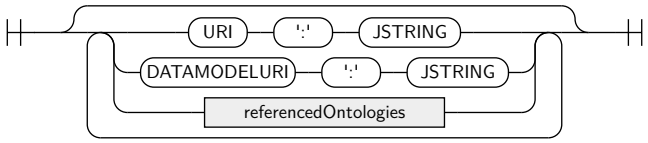
Die folgenden Syntax-Diagramme beschreiben die Syntax der *mimesis.variants.DSL*, wie sie im Rahmen der Implementierung zur Evaluation umgesetzt wurde.

Da zum Zeitpunkt der Entwicklung die konkrete Modellierung einiger Aspekte noch nicht final festgelegt war, finden sich z.T. noch generische Konstrukte in der Grammatik. Die Syntaxbeschreibung ist daher in dieser Form nicht zur Definition eines Standards geeignet und wird in zukünftigen Versionen finalisiert.

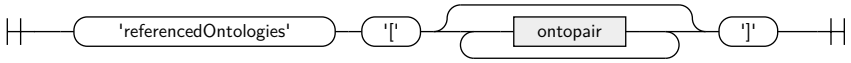
variant_description:



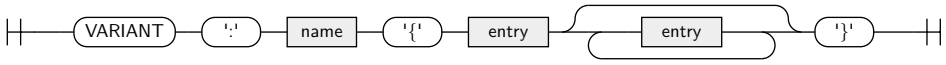
metadata:



referencedOntologies:

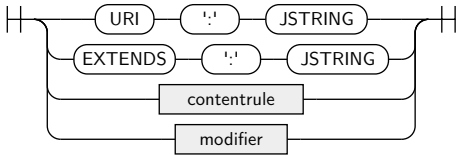


variant:

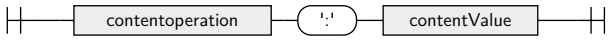


name:

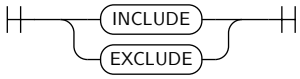
entry:



contentrule:

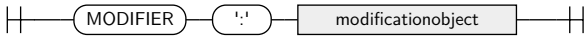


contentoperation:

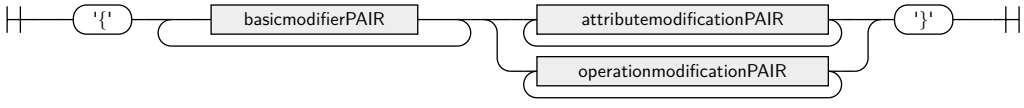


contentValue:

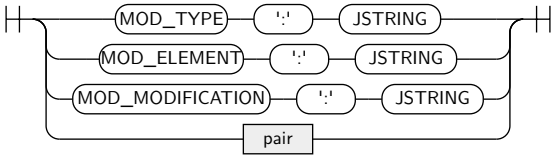
modifier:



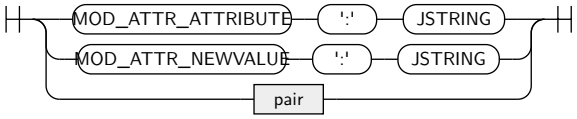
modificationobject:



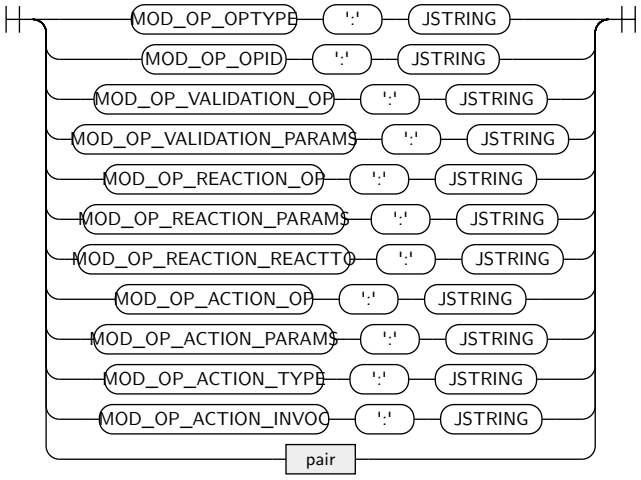
basicmodifierPAIR:



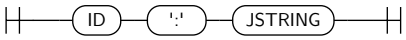
attributemodificationPAIR:



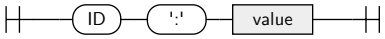
operationmodificationPAIR:



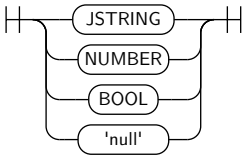
ontopair:



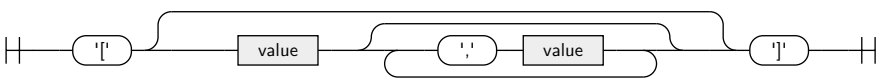
pair:



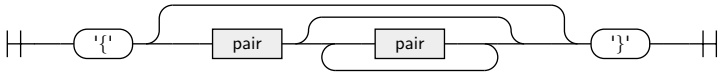
value:



valuearray:



object:



Terminale

VARIANTDESCRIPTION	variant_description	**MOD_ATTR_ATTRIBUTE**	attribute
VARIANT	variant	**MOD_ATTR_NEWVALUE**	newvalue
DATAMODELURI	datamodelURI	**MOD_OP_OPTYPE**	operationType
URI	uri	**MOD_OP_OPID**	operationID
EXTENDS	extends	**MOD_OP_VALIDATION_OP**	validation_operation
		MOD_OP_VALIDATION_PARAMS	validation_parameters
INCLUDE	include	**MOD_OP_REACTION_OP**	reaction_operation
EXCLUDE	exclude	**MOD_OP_REACTION_PARAMS**	reaction_parameters
		MOD_OP_REACTION_REACTTO	reaction_reactto
MODIFIER	modify	**MOD_OP_ACTION_OP**	action_operation
MOD_TYPE	type	**MOD_OP_ACTION_PARAMS**	action_parameters
MOD_ELEMENT	element	**MOD_OP_ACTION_TYPE**	action_type
MOD_MODIFICATION	modification	**MOD_OP_ACTION_INVOC**	action_invocation

A.3.3 Benutzungsschnittstellenmodell als DSL (mimesis.ui.DSL)

Das Benutzungsschnittstellenmodell dient sowohl zur Beschreibung der abstrakten (AUI-Modell) als auch der konkreten Benutzungsschnittstelle (CUI-Modell). Das AUI-Modell wird initial aus dem Anwendungsmodell abgeleitet und in weiteren Transformationsschritten zu einer konkreten Benutzungsschnittstelle verfeinert. Das Modell beschreibt die Struktur und Interaktionselemente, abstrahiert jedoch noch von konkreten Darstellungselementen.

Die Beschreibung des Benutzungsschnittstellenmodells folgt dem in Anhang A.2.3 zusammengefassten Metamodell und stellt eine textuelle Form der darin enthaltenen Informationen dar.

Aufbau der Beschreibung. In Abbildung A.18 ist die Grundstruktur der Beschreibung dargestellt. Die Beschreibung wird durch einen Metadatenbereich eingeleitet gefolgt von einer Liste von Darstellungseinheiten (*displayunits*). Die Beschreibung einer Darstellungseinheit erfolgt in vier Bereichen.

- Attribute
- Darstellungselemente
- Aktionen
- Untereinheiten

Der Bereich *Attribute* enthält Angaben zur Darstellungseinheit selbst (vgl. Tabelle A.9, *DisplayUnit*). Der Bereich *Darstellungselemente* enthält eine Liste der Darstellungs- und Interaktionselemente der Darstellungseinheit (*displayelements*). Dies können Gruppen oder Felder der Benutzungsschnittstelle sein (s. unten), die auf der Darstellungseinheit präsentiert werden. Der Bereich *Aktionen* enthält eine Liste von Aktionen, die mit der Darstellungseinheit assoziiert sind (z.B. das Senden des Inhalts an ein Backend. o.ä.). Im Bereich *Untereinheiten* sind weitere Darstellungseinheiten angegeben (*displayunits*), die der Darstellungseinheit untergeordnet sind (z.B. einer Seite zugeordnete Unterseiten in grafischen Benutzungsschnittstellen).

Abbildung A.18. Struktur des Benutzungsschnittstellenmodells

Darstellungselemente. Die Abbildungen A.19 (a) und A.19 (b) zeigen exemplarisch den Aufbau der Beschreibung von Darstellungselementen für *Gruppen* und *Felder*. Beide Darstellungselemente werden in einem *Attribut*-Bereich spezifiziert. In Tabelle A.9, *DisplayElement* sind die Attribute zusammengefasst, die Gruppen und Felder gemeinsam aufweisen. Neben diesen Attributen enthalten beide Darstellungselemente die Angabe zu assoziierten *Aktionen, Reaktionen* und *Validierungen* (s. unten).

Gruppen besitzen zusätzlich einen Bereich (*Elemente*), der eine Liste der Darstellungselemente enthält, die in der Gruppe zusammengefasst werden sollen (*displayelements*). Dies können wiederum Gruppen und Felder sein. *Felder* besitzen zusätzliche Attribute zur Beschreibung ihrer Typ-Eigenschaften und Beziehung zum zugrunde gelegten Datenmodell im Anwendungskontext (vgl. Tabelle A.9, *Field* und Tabelle A.10).

Aktionen, Reaktionen und Validierungen. Abbildung A.20 zeigt die Beschreibungen für Aktionen, Reaktionen und Validierungen, die für Gruppen und Felder angegeben werden können. Die Beschreibung erfolgt anhängig von der Operationsart anhand der Attribute aus Tabelle A.11. Für jedes Darstellungselement kann in dem entsprechenden Bereich (vgl. Abbildung A.19 (a) bzw. A.19 (b) eine Liste der dargestellten Beschreibungen angegeben werden.

*im Prototyp nicht umgesetzt

(a) Beschreibung einer Gruppe

(b) Beschreibung eines Feldes

Abbildung A.19. Beschreibung von Darstellungselementen

Aktion

```
{
 "actionId"    :"fn_liab.sendData_id",
 "actionLabel" :"Daten senden",
 "actionName"  :"fn_liab.SendData",
 "actionParams":"",
 "actionType"  :"element",
 "actionInvoke":"click"
}
```

Reaktion

```
{
 "reactionId"      :"fn_liab.prefillCity_id",
 "reactionName"    :"fn_liab.prefillCity",
 "reactionParams"  :"liability.customerdata.address.zip,
                           $liability.customerdata.address.city",
 "reactionReactTo" :"liability.customerdata.address.zip"
}
```

Validierung

```
{
 "validatorId"     :"validation.checkChildrenAmount_id",
 "validatorName"   :"validation.checkChildrenAmount",
 "validatorParams" :"liability.customerdata.children.numberOfChildren"
}
```

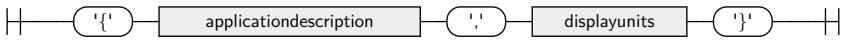
Abbildung A.20. Beschreibung von Aktionen, Reaktionen und Validierungen

Syntax mimesis.ui.DSL

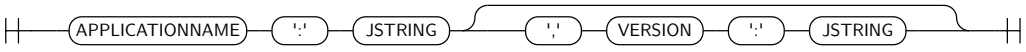
Die folgenden Syntax-Diagramme beschreiben die Syntax der *mimesis.ui.DSL*, wie sie im Rahmen der Implementierung zur Evaluation umgesetzt wurde.

Da zum Zeitpunkt der Entwicklung die konkrete Modellierung einiger Aspekte noch nicht final festgelegt war, finden sich z.T. noch generische Konstrukte in der Grammatik. Die Syntaxbeschreibung ist daher in dieser Form nicht zur Definition eines Standards geeignet und wird in zukünftigen Versionen finalisiert.

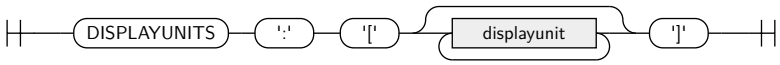
uidescription:



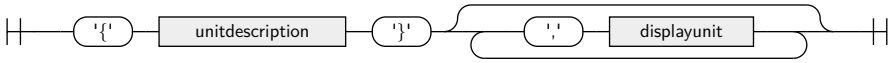
applicationdescription:



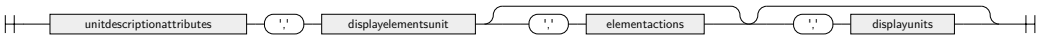
displayunits:



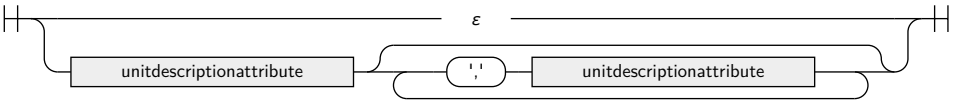
displayunit:



unitdescription:



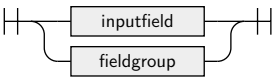
unitdescriptionattributes:



displayelementsunit:

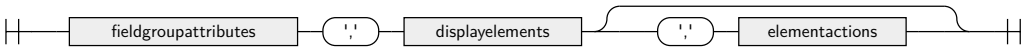
displayelements:

displayelement:

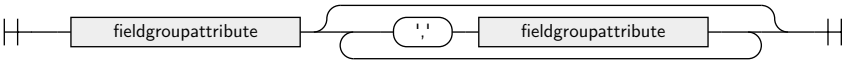


fieldgroup:

fieldgroupdescription:

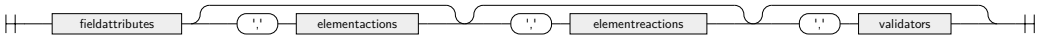


fieldgroupattributes:

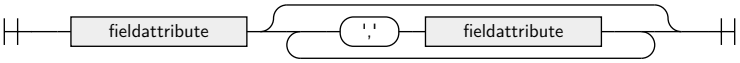


inputfield:

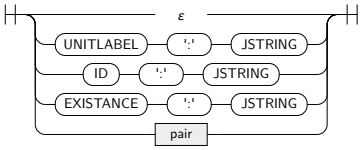
inputfielddescription:



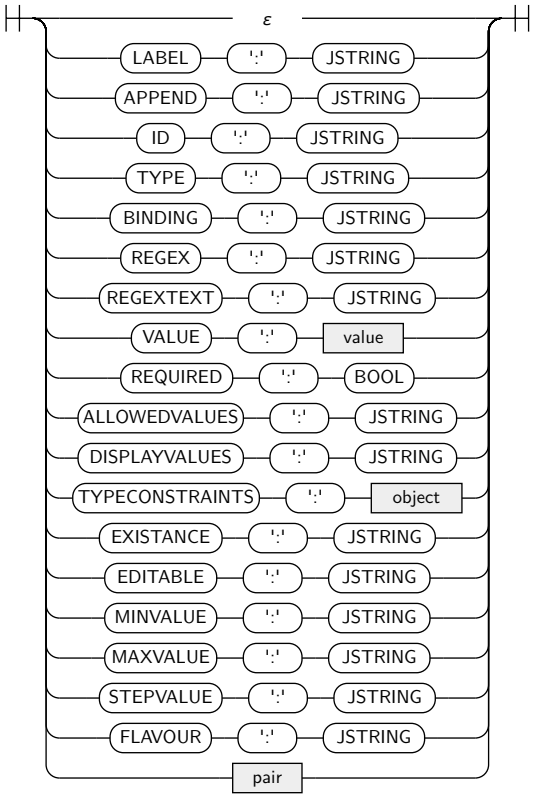
fieldattributes:



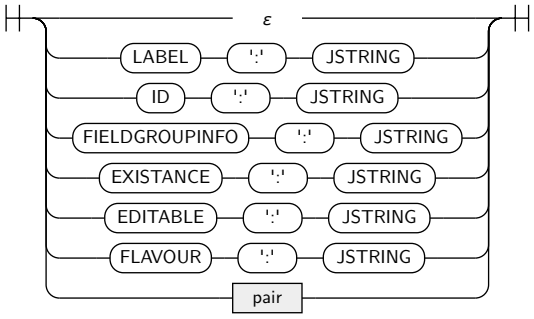
unitdescriptionattribute:



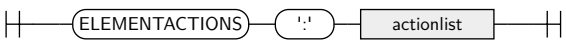
fieldattribute:



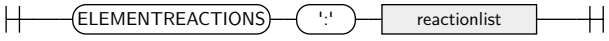
fieldgroupattribute:



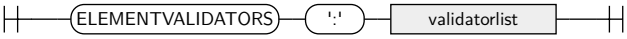
elementactions:



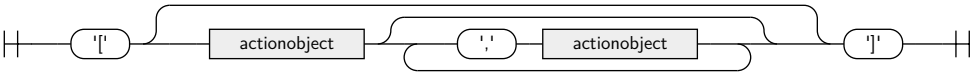
elementreactions:



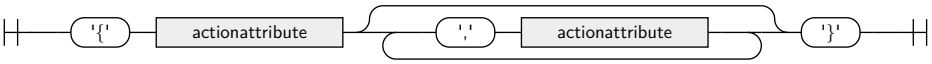
validators:



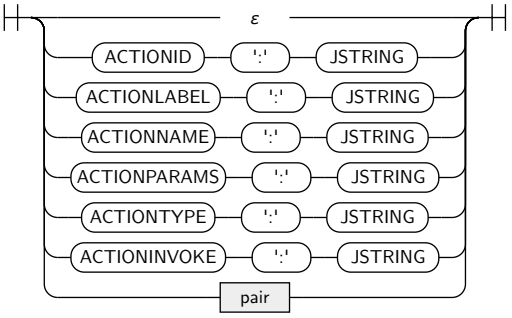
actionlist:



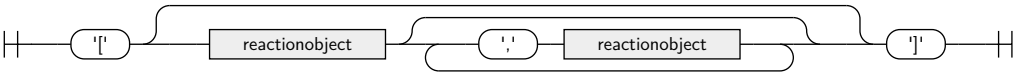
actionobject:



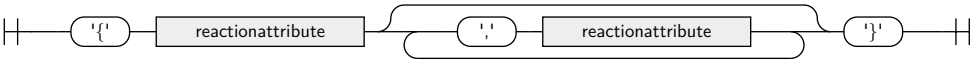
actionattribute:



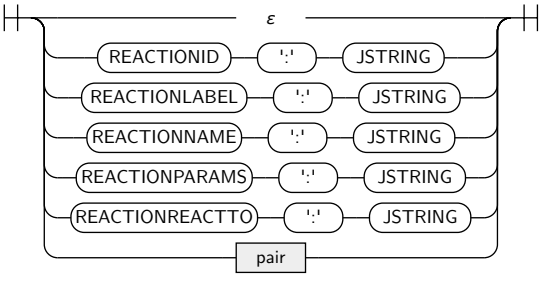
reactionlist:



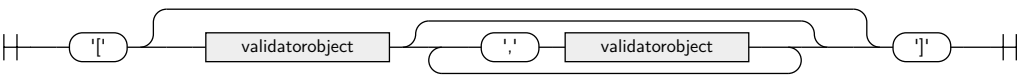
reactionobject:



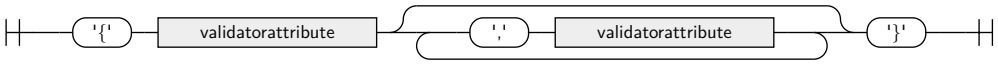
reactionattribute:



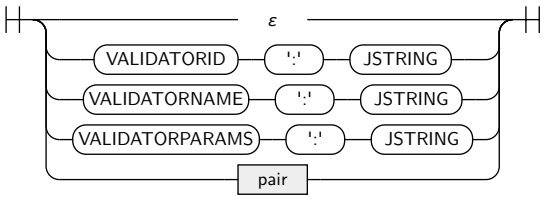
validatorlist:



validatorobject:

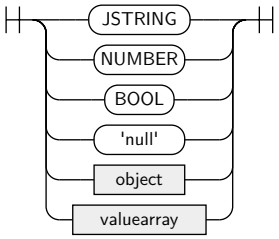


validatorattribute:

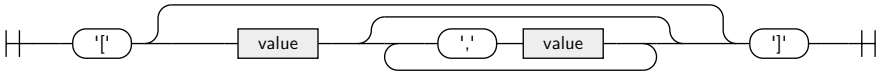


pair:

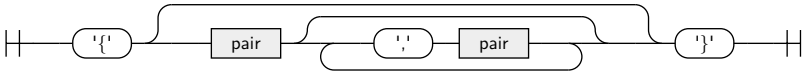
value:



valuearray:



object:



Terminale

APPLICATIONNAME	applicationName	**MINVALUE**	min	**REACTIONID**	reactionID
VERSION	_version	**MAXVALUE**	max	**REACTIONLABEL**	reactionLabel
DISPLAYUNITS	displayunits	**STEPVALUE**	steps	**REACTIONNAME**	reactionName
DISPLAYELEMENTS	displayelements	**ELEMENTACTIONS**	actions	**REACTIONPARAMS**	reactionParams
UNITLABEL	unitlabel	**ELEMENTREACTIONS**	reactions	**REACTIONREACTTO**	reactionReactTo
LABEL	label	**ELEMENTVALIDATORS**	validators		
ID	id	**FIELDGROUPINFO**	fieldgroupInfo	**ACTIONID**	actionID
TYPE	type	**TYPECONSTRAINTS**	typeConstraints	**ACTIONLABEL**	actionLabel
FLAVOUR	flavour	**VALIDATION**	validation	**ACTIONNAME**	actionName
BINDING	binding	**VALIDATIONERROR**	validationError	**ACTIONPARAMS**	actionParams
VALUE	value	**REGEX**	regEx	**ACTIONTYPE**	actionType
EXISTANCE	exists	**REGEXTEXT**	regExText	**ACTIONINVOKE**	actionInvoke
EDITABLE	editable	**REQUIRED**	required		
		ALLOWEDVALUES	allowedValues	**VALIDATORID**	validatorID
		DISPLAYVALUES	valueLabels	**VALIDATORNAME**	validatorName
		APPEND	append	**VALIDATORPARAMS**	validatorParams

A.4 Algorithmen zur Herleitung des Variantenmodells

Dieser Anhang beschreibt die Algorithmen zur Herleitung des Variantenmodells aufbauend auf dem Anwendungsmodell und der Variantenbeschreibung im Detail.

A.4.1 Anwendung der Elementregeln (applyElementrule)

Die Anwendung einer Elementregel bestimmt die Existenz eines Knotens im Strukturbaum der Variante und damit seine Sichtbarkeit. Zur Kennzeichnung der Exklusion eines Knotens werden Markierungen verwendet. Ist ein Knoten markiert, so soll er exkludiert werden. Besitzt ein Konten keine Markierung, ist er sichtbar. Für den markierten Strukturbaum sollen folgende Grundregeln gelten:

1. Jeder sichtbare Knoten des Baums muss vom Wurzelknoten aus sichtbar sein. D.h. er muss auf einem Pfad des Strukturbaums liegen, auf dem alle Elternknoten ebenfalls sichtbar sind.
2. Jeder innere Knoten ist genau dann sichtbar, wenn mindestens ein sichtbares Blatt in einem untergeordneten Teilbaum existiert.

Bei Änderung der Markierung eines Knotens im Baum müssen daher ggf. auch Kind- bzw. Elternknoten angepasst werden, um diese Regeln zu erfüllen. Daraus lassen sich für die Statusänderung eines Knotens Aktionen angeben, die den Zustand des Strukturbaums gemäß den Regeln konsistent halten:

- Wird ein Knoten markiert (Exklusion des Knotens), werden auch alle ggf. vorhandenen Unterknoten markiert. Besitzt dadurch der Elternknoten keine unmarkierten Elemente mehr, wird auch dieser markiert.
- Wird die Markierung eines Knotens entfernt (Inklusion des Knotens), werden auch alle Markierungen der Unterknoten entfernt. Damit der Knoten sichtbar wird, müssen zudem ggf. vorhandene Markierungen von Elternknoten auf dem Pfad zur Wurzel entfernt werden.

Algorithmus 6 zeigt die Durchführung einer Exklusion bzw. Inklusion, der in Form von Pseudocode diese Operationen beschreibt. Zuerst wird das betroffene Element des Strukturbaums und die durchzuführende Operation bestimmt.

Handelt es sich bei der Elementregel um eine **Exklusion** des Elementknotens, wird dieser markiert. Zudem werden alle Unterknoten ebenfalls markiert (*excludeAllChildren()*). Diese Exklusion erfolgt rekursiv, sodass am Ende die Elemente des gesamte Teilbaums unter dem Elementknoten markiert sind. Anschließend wird geprüft, ob durch die Exklusion Elternknoten auf dem Weg zur Wurzel existieren, die durch die Exklusion keine unmarkierten Unterknoten mehr besitzen und diese ebenfalls markiert (*excludeEmptyParentsOnPathToRoot()*).

Zur **Inklusion** eines Elementknotens, wird eine ggf. vorhandene Markierung entfernt. Die Inklusion wird rekursiv für alle Unterknoten durchgeführt (*includeAllChildren()*). Da die Elternknoten ggf. durch vorangegangene Elementregeln markiert wurden, müssen diese wieder inkludiert werden. Ausgehend vom Elementknoten wird hierzu bei allen inneren Knoten auf

Algorithmus 6 Anwendung der Elementregeln

```
 1: procedure APPLYELEMENTRULE
 2:     inputs
 3:         Elementregel (r)
 4:         → r.element: betroffenes Modellelement
 5:         → r.type: include / exclude
 6:         Variantenmodell (AM'),
 7:     result modifiziertes Variantenmodell (AM')
 8:     begin
 9:         element ← AM'.lookup(r.element)
10:         if r.type == exclude then
11:             true → element.exclusionMarker
12:             AM'.excludeAllChildren (element);
13:             AM'.excludeEmptyParentsOnPathToRoot (element);
14:         if r.type == include then
15:             false → element.exclusionMarker
16:             AM'.includeAllChildren(element);
17:             AM'.includeParentsOnPathToRoot(element);
```

dem Weg zur Wurzel des Strukturbaums eine ggf . vorhandene Markierung entfernt (*includeParentsOnPathToRoot()*).

Das Ergebnis der Operation ist ein modifizierter Strukturbaum, dessen Knoten ausgehend von der Wurzel sichtbar sind.

A.4.2 Anwendung der Elementmodifikatoren

Elementmodifikatoren ersetzen, ergänzen oder entfernen Eigenschaften eines Strukturbaumknotens. In Abschnitt 6.4 wurden folgende Modifikatoren aufgeführt:

- *OperationModifier*: Änderung des Verhaltens der Anwendung (z.B. Existenz, Validierung, elementinitiierte Operationen)
- *AttributeModifier*: Änderung der Attribute eines Datenelements
- *SemanticTagModifier*: Änderung der SeamticTags eines Elements
- *StructureModifier*: Änderung der Struktur der Anwendungsbeschreibung

Jede dieser Modifikatoren enthält alle Informationen zur Art der Modifikation (z.B. Setzen oder Löschen) und den neuen Zustand der Eigenschaft. Algorithmus 7 zeigt exemplarisch das Vorgehen zur Modifikation für *OperationModifier* und *AttributeModifier*.

Die Eingabe für den Algorithmus ist der Modifikator und das Anwendungsmodell. Zuerst wird das betroffene Element des Strukturbaums und der Typ des Modifikators bestimmt. Der Algorithmus zeigt dies für *OperationModifier* und *AttributeModififer*. Anhand des Typs und der im Modifier enthaltenen Informationen wird die Operation bestimmt, die auf dem Baumknoten des Elements ausgeführt werden soll. Ab Zeile 9 wird z.B. für den *OperationModifier* bestimmt, ob die Operation *hinzugefügt, geändert* oder *entfernt* werden soll und entsprechende Operation mit den neuen Werten für die Änderung des Elementknotens gerufen. Analog wird für weitere Modifikatortypen verfahren. Für jeden Modifikator werden hierzu entsprechende Operationen ausgeführt, die den Elementknoten modifizieren.

Algorithmus 7 Anwendung der Elementmodifikatoren

```
1:  procedure APPLYELEMENTMODIFICATON(m, AM)
2:      inputs
3:          ElementModifikator (m)
4:          → m.element: betroffenes Modellelement
5:          → m.modification: ElementModifier
6:          Anwendungsmodell (AM),
7:      begin
8:          element ← AM.lookup(r.element)
9:          if m ofType OperationModifier then
10:             if m.modification.type == insert then
11:                 element.insertOperation (m.modification.ID,m.modification)
12:             if m.modification.type == remove then
13:                 element.removeOperation (m.modification.ID)
14:             if m.modification.type == replace then
15:                 element.replaceOperation (m.modification.ID, m.modification)
16:         if m ofType AttributeModifier then
17:             element.setAttribute (m.modification.key, m.modification.newValue)
18:         if m ofType ... then
19:             ...
```

Das Ergebnis ist ein Strukturbaum, dessen Knoten gemäß der Regeln modifiziert sind, die in der Variantenbeschreibung angegeben wurden.

A.5 Ontologische Beschreibung von Anwendungsmodellen

Im Folgenden werden die in Abschnitt 9.4 skizzierten Vorgehensweisen zur Beschreibung des Anwendungsmodells detailliert. Abschnitt A.5.1 beschreibt die Repräsentation des Anwendungsmodells in Form einer RDF/OWL-Ontologie. Abschnitt A.5.2 beschreibt die Erweiterung dieser Ontologie um ein Annotations-Profil zur Beschreibung der Datenanbindung. Abschnitt A.5.3 stellt abschließend einen Algorithmus vor, der aus einer Applikations-Ontologie-Instanz eine Dienst-Ontologie-Instanz zur Laufzeit erzeugt.

A.5.1 Anwendungsmodell als RDF/OWL-Ontologie

Die Repräsentation des Anwendungsmodells als RDF/OWL-Ontologie unterliegt den in den Abschnitten 5.3-5.5 formulierten Anforderungen hinsichtlich Struktur, Typisierung der Ein-/Ausgabeelemente und des Verhaltens. Das in Abschnitt 6.3 aus diesen Anforderungen abgeleitete Metamodell beschreibt Benutzungsschnittstellen als attribuierten Strukturbaum mit Gruppen, Daten, Beziehungen und deren Eigenschaften. Dieser Abschnitt beschreibt die Abbildung der im Metamodell enthaltenen Informationen auf eine RDF/OWL-Ontologie.

Tabelle A.12 fasst die in den Abschnitten 5.3-5.5 mit (IR.1-IR.14) bezeichneten Anforderungen an den Informationsgehalt des Anwendungsmodells zusammen[56]. In der Spalte *Metamodell* ist angegeben, mit welchen Modellelementen die Anforderungen im Metamodell umgesetzt wurden (vgl. Anhang A.2.1, Abbildung A.6). Zur Überführung des im Metamodell beschriebenen Strukturbaums in einen RDF-Graphen müssen diese Konstrukte mit RDF/OWL-Mitteln beschrieben werden.

ID	Information	im Metamodell	OWL-Mapping
Modellierung Struktur / Aufbau			
(IR.1)	**Gruppierung zusammenhängender Elemente Innere Kohäsion**	DataGroup Relation	`OWLClass` `OWLObjectProperty`
	Äußere Kohäsion	DataGroup Attribut	`OWLAnnotation`
(IR.2)	**Temporale Abfolge der Gruppen- und Datenelemente**	geordnete Relation	`OWLAnnotation`
Modellierung typisierte Ein-/Ausgabe			
(IR.5)	**Typ des zu erfassenden Datums**	DataItem + Typ Attribut	`OWLDataProperty`
	Spezifikation geltender Restriktionen	DataItem Attribute	`OWLAnnotation`
(IR.6)	**Fachliche Semantik des Datums**	DataItem Attribut	`OWLAnnotation`
(IR.7)	**Fachliche Semantik der Gruppe**	GroupItem Attribut	`OWLAnnotation`
ModellierungVerhalten			
(IR.10)	**Sicht-/Editierbarkeit von Informationen**	DescriptionElement Attribut	`OWLAnnotation`
(IR.11)	**Validierung eingegebener Daten**	DescriptionElement Attribut	`OWLAnnotation`
(IR.12)	**Reaktionen auf Änderungen im Kontext**	DescriptionElement Attribut	`OWLAnnotation`
(IR.13)	**Elementinitiierte Aktionen**	DescriptionElement Attribut	`OWLAnnotation`
(IR.14)	**Nutzer- und Systeminitiierte Aktionen**	DescriptionElement Attribut	`OWLAnnotation`

Tabelle A.12. Abbildung der Metamodellelemente auf OWL-Konstrukte

[56] Auf eine Darstellung der Anforderungen für inhaltliche und technische Varianten wird an dieser Stelle verzichtet, da diese lediglich zur Erstellung eines Variantenmodells bzw. bei der Generierung relevant sind.

Insbesondere Struktur und Aufbau des Anwendungsmodells lassen sich mit Basis-Elementen von RDF/OWL beschreiben. Jedoch können nicht alle im Metamodell enthaltenen Informationen auf RDF/OWL-Standardelemente abgebildet werden. Ontologien sind zur Repräsentation von Faktenwissen konzipiert. Daher enthält RDF/OWL keine Elemente zur Beschreibung von nutzungsspezifischen Informationen wie der Sequenz von Daten (IR.2) oder dem Verhalten (IR.10-IR.14). Zur Beschreibung dieser Eigenschaften werden Informationen benötigt, die erst zur Laufzeit vorhanden sind (Instanzdaten), z.B. zum Ein-/Ausblenden von Fragen aufgrund bereits getätigter Eingaben. Nach meinem besten Wissen enthält RDF/OWL zudem kein standardisiertes Konzept zur deklarativen Beschreibung von Operationen oder zur Modellierung von Bedingungen, die auf Instanzdaten der beschriebenen Ontologie basieren[57].

Zur näheren Beschreibung von Entitäten, Datenelementen und Relationen bietet OWL das Konzept der **Annotationen**[58]. Mit Annotationen können OWL-Elemente um Informationen ergänzt werden, die bei der Verarbeitung der Ontologie optional genutzt werden können. Annotationen können in sog. **Profilen** zusammengefasst werden. Ein Profil ist eine Sammlung von Annotationen für einen bestimmten Problembereich und beschreibt die Semantik der enthaltenen Annotationen. Ontologien können Profile referenzieren und damit um semantisch bestimmte Informationen ergänzt werden. Die Verwendung von Annotationen führt zu einer **annotierten Ontologie** (*profiled ontology*) und ermöglicht die Nutzung einer Ontologie in spezifischen Problembereichen[59].

Die im Folgenden dargestellte Lösung verwendet **annotierte OWL-Ontologien** zur erweiterten Beschreibung des Strukturbaums der Benutzungsschnittstelle. Der Strukturbaum selbst wird als RDF-Graph mit Standard-OWL-Mitteln repräsentiert, dessen Knoten über Annotationen eines Basisprofils mit zusätzlichen Informationen angereichert werden. In Tabelle A.12, Spalte *OWL-Mapping* sind die OWL-Elemente aufgeführt, die zur Abbildung der Metamodellelemente auf OWL-Konstrukte verwendet werden. Die Umsetzung wird in den folgenden Abschnitten dargestellt.

Repräsentation des Strukturbaums als RDF-Graph

Zur Modellierung der Gruppen- und Datenelemente sowie deren Beziehungen bietet OWL Konstrukte. Damit kann die Struktur der Anwendung abgebildet werden. Abbildung A.21 zeigt die Projektion strukturellen Elemente des Metamodells auf die Elemente einer OWL-Ontologie. Zusätzlich sind konkrete Beispiele aus der *Antragstrecke Haftpflichtversichering* in *Turtle*-Notation[60] dargestellt.

[57] Liu et al. [133] schlagen eine Lösung zur Modellierung von Existenzbedingungen in Form von Regeln in einer Wissensdatenbank vor, auf die ein *Reasoner* zur Laufzeit einer Anwendung zugreifen kann. In Bosch et al. [16] wird ein Ansatz vorgeschlagen, der diese Informationen über SPARQL-Anfragen herleitet. Beide Ansätze erfordern allerdings eine zur Laufzeit vorhandene Schulssfolgerungs-Komponente (Reasoner), die den aktuellen Zustand, die Regeln bzw. Anfragen enthält. Eine solche Komponente existiert jedoch nicht in allen Laufzeitumgebungen von Benutzungsschnittstellen (z.B. in webbasierten Anwendungen, die in einem Webbrowser ausgeführt werden).

[58] https://www.w3.org/TR/owl-ref/#Annotations

[59] In den Arbeiten von Khushraj et al. [115]) und Gaulke et al. [76] werden annotierte Ontologien zur Beschreibung von Benutzungsschnittstellen angewendet. Khushraj et al. nutzen Annotationen zur Erweiterung von OWL-basierten Schnittstellenbeschreibungen für WebServices um Benutzungsschnittstellenspezifika. In Galuke et al. wird der Ansatz verwendet, um ein Domain/Task-Modell um Informationen zur Benutzungsschnittstelle zu erweitern.

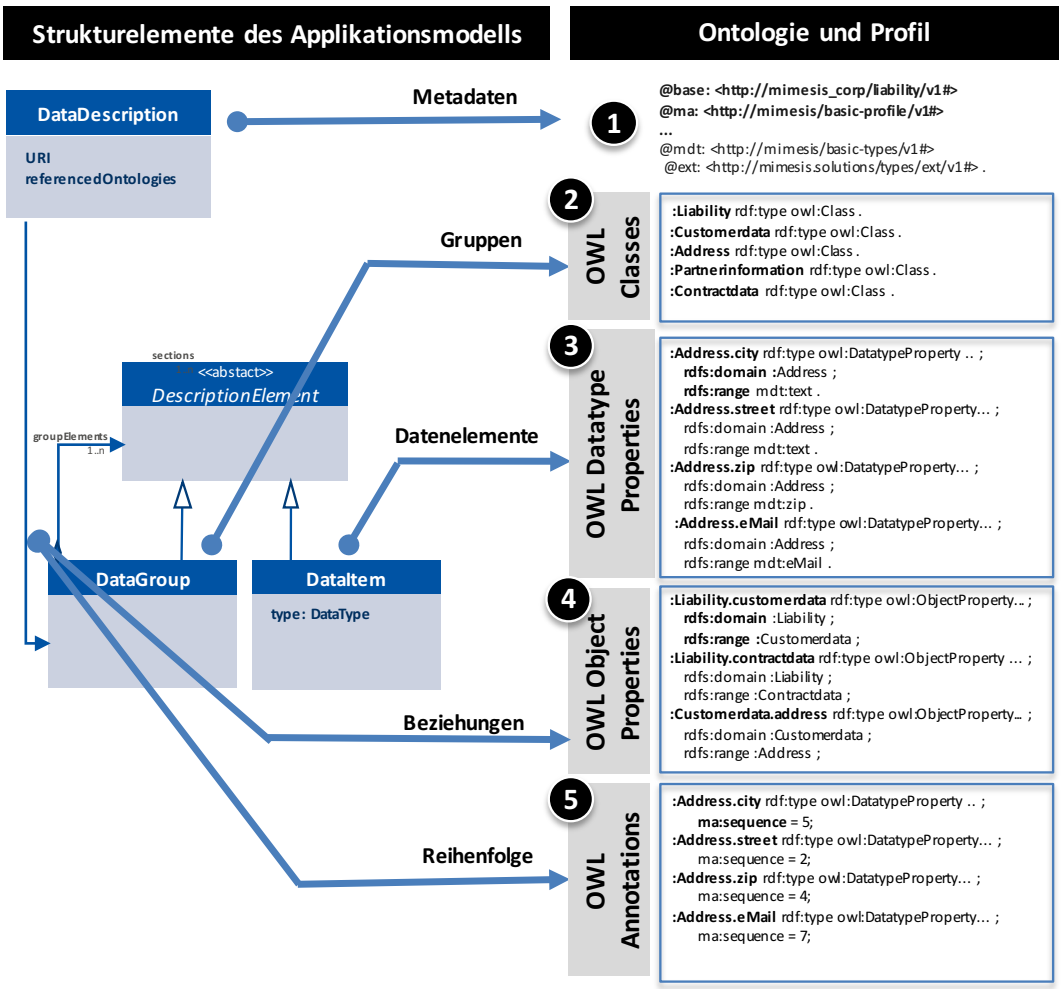


Abbildung A.21. Projektion der Struktur auf OWL-Konstrukte

Die Grundinformationen der Ontologie (z.B. URI, referenzierte Ontologien) entsprechen den Metadaten des Anwendungsmodells ① und werden entsprechend in der Präambel der Ontologie angegeben. Die Beschreibung von Gruppen des Anwendungsmodells (**DataGroups**) erfolgt über Entitäten der Ontologie ②(**owl:Class**) (z.B. *:CustomerData*, *:Address*, *:PartnerInformation* und *:ContractData*). Die Datenelemente (**DataItems**) einer Gruppe werden als **owl:DatatypeProperties** beschrieben ③. Sie sind einer Entität zugeordnet (*rdfs:domain*) und mit den grundlegenden Typinformationen versehen (*rdf:range*), die auch im Metamodell enthalten sind (*type*-Attribut). Die hierarchischen Beziehungen zwischen Gruppen werden über **owl:ObjectProperties** beschrieben. In Abbildung A.21, ④ werden beispielsweise *:Customerdata* und *:ContractData* als Teil von *:Liability* festgelegt.

Mit diesem Mapping werden die Informationen zur Gruppierung und inneren Kohäsion (IR.1) sowie grundlegende Typinformationen der Datenelemente (IR.5) mit RDF/OWL-Basiskonzepten beschrieben. Es entsteht **ein dem Strukturbaum äquivalenter RDF-Graph**, dessen innere Knoten den Gruppen und dessen Blätter den Datenelementen entsprechen.

60 https://www.w3.org/TeamSubmission/turtle/

Die **Sequenz der Gruppen und Datenelemente** lässt sich hingegen nicht mit diesen Standardmitteln beschreiben. Hierzu werden Annotationen verwendet ⑤, die im Folgenden beschrieben werden.

Annotationen und Profil zur Detaillierung des Anwendungsmodells

Die Beschreibung der weiteren Aspekte des Anwendungsmodells wird im Metamodell durch Attribuierung der Knoten für Gruppen und Datenelemente erreicht. Die Knoten des RDF-Graphen können analog attribuiert werden, indem den **Attributen entsprechende Annotationen** verwendet werden.

Tabelle A.13 zeigt die Annotationen, die das **Basisprofil zur Beschreibung eines Anwendungsmodells** bilden. Sie sind nach den Themenbereichen *Struktur/Aufbau*, *Typisierte Ein-/Ausgabe* und *Verhalten* gruppiert, zu deren Beschreibung sie beitragen. Zudem ist dargestellt, welche Elemente des RDF-Graphen damit annotiert werden können und welchen Metamodell-Attributen sie entsprechen (vgl. Tabellen A.3-A.7).

Annotation	Geltungsbereich	Entsprechung im Metamodell	Anmerkung
Aufbau Struktur			
:sequence	`owl:Class` `owl:DataProperty`	geordnete Liste von DataGroups und DataItems	Eplizite Bestimmung der Position des Elements im Fragefluss der es umgebenden Hierarchiestufe.
:cohesion	`owl:Class`	Attribut: DataGroup.cohesion	Innere/Äußere Kohäsion analog Metamodell: {weak, regular, strong}
Typinformationen			
:cardinality	`owl:Class` `owl:DataProperty`	Attribut: DescriptionElement.cardinality	Multiplizität einer Gruppe bzw. Elementen in einer Auswahl
:type	`owl:DataProperty`	Attribut: DataItem.type	Basistyp analog Metamodell
:<Restriktion>, ***z.B :restrictedTo,*** ***:initialvalue,*** ***:min, :max***	`owl:DataProperty`	Relation zu Restrictions: DataItem.restrictions	Einschränkungen der Daten analog Metamodell. Modellierung in Ontologie analog DSL als Attribut/Wert-Repräsentation
:sematicTags	`owl:Class` `owl:DataProperty`	Relation zu SemanticTags: DescriptionElement.semanticTags	Semantische Tags analog Metamodell. Syntax: <namespace>:<tag>*
Verhalten			
:existIf **:activeIf**	`owl:Class` `owl:DataProperty`	Relation zu ModelCondition: DescriptionElement.existsIf	Beschreibung als Bedingung auf Instanzdaten in einem Feld. Analog Umsetzung in DSL
:validations	`owl:Class` `owl:DataProperty`	Relation zu Validation: DescriptionElement.validations	Verwendung einer kompakte Darstellung analog DSL Syntax: [<trigger>:<operation>(<parameter >*)]*
:reactions	`owl:Class` `owl:DataProperty`	Relation zu Reaction: DescriptionElement.reactions	Verwendung einer kompakte Darstellung analog DSL Syntax: [<element>:<operation>(<parameters>*)]*
:actions	`owl:Class` `owl:DataProperty`	Relation zu Action: DescriptionElement.actions	Verwendung einer kompakte Darstellung analog DSL Syntax: [<type>:<trigger>:<operation>(<parameter>*)]*

Tabelle A.13. Annotationen des Basisprofils

Die Annotationen entsprechen weitgehend den Angaben, die auch zur Beschreibung des Modells als DSL verwendet wurden (s. Abschnitt A.3.1). Der Unterschied besteht in der Hinzunahme einer Annotation zur Beschreibung der temporalen Abfolge (Tabelle A.13, *:sequence*). Im Metamodell wird diese *implizit* über eine geordnete Liste bzw. in der DSL über die Reihenfolge der Elemente in der Beschreibung abgebildet. Da RDF keinen geordneten Graphen beschreibt, wird diese Eigenschaft hier explizit über eine Annotation angegeben.

Abbildung A.22 zeigt exemplarisch die Anwendung der Annotationen. Zur Vervollständigung der Beschreibung zur **Struktur** wird die **temporale Abfolge** der Gruppen und Elemente durch Angabe einer *:sequence*-Annotation bestimmt. Diese wird *owl:Class* und *owl:DatatypeProperties* angehängt und bestimmt die relative Position des Elements auf seiner Hierarchiestufe. Abbildung A.22, ① zeigt die Festlegung der Reihenfolge exemplarisch für die Datenelemente *Postleitzahl* und *Ort*, die Teil der Adressdaten sind. Die Beschreibung der **äußeren Kohäsion** einer Gruppe zu ihrem Umfeld erfolgt über eine *:cohesion*-Annotation analog dem Attribut des Metamodells ②.

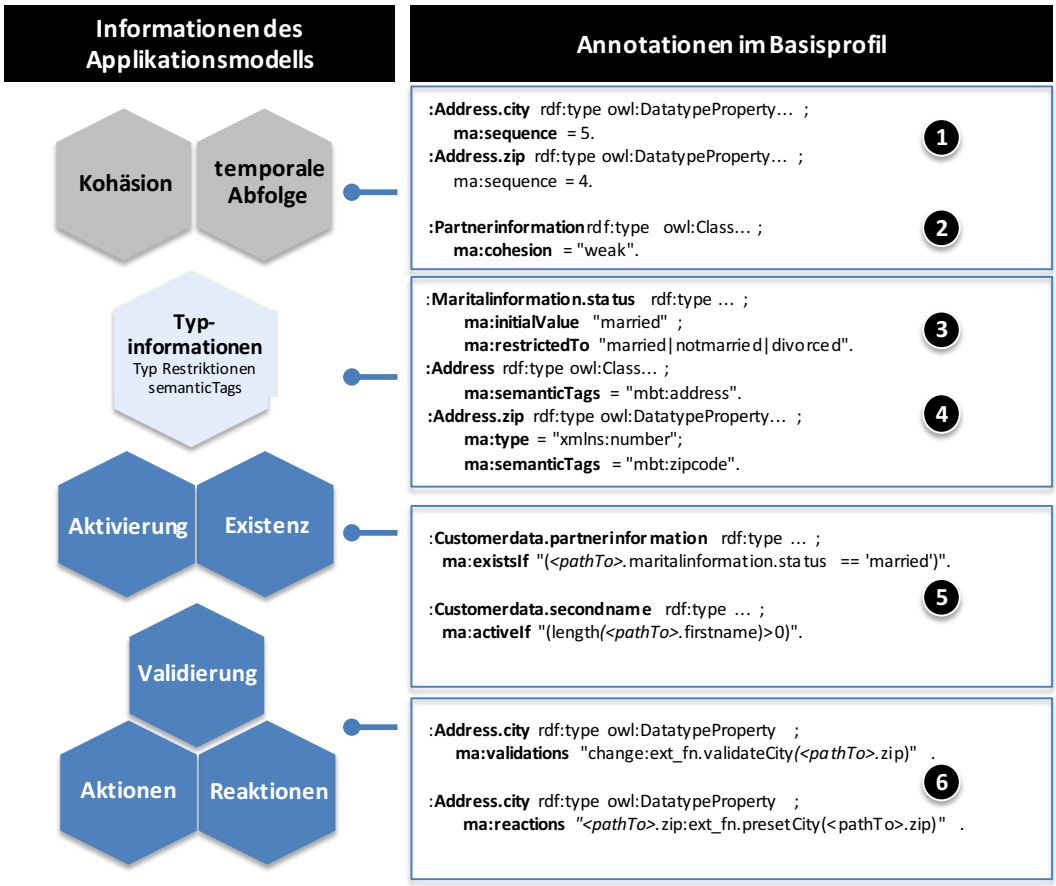


Abbildung A.22. Projektion der Struktur, Typinformationen und des Verhaltens über Annotationen

Typinformationen (Typ, Restriktionen, semantische Annotationen) wurden im Anwendungsmodell über Attribute der Klassen *DescriptionElement* und *DataItem* beschrieben. Analog können *owl:Class* bzw. *owl:DatatypeProperty*-Elemente im RDF-Graphen mit Typ-Annotationen versehen werden. Der **Typ** wird über die *:type*-Annotation für das *owl:DatatypeProperty*-Element angegeben, der das Datenelement repräsentiert. Die möglichen Ausprägungen entsprechen den für das Metamodell festgelegten Werten (vgl. Anhang A.2.1, Tabelle A.6). Für das Datenelement geltende **Restriktionen** werden analog über Annotationen beschrieben, die ebenfalls im Rahmen des Metamodells festgelegt wurden (Anhang A.2.1, Tabelle A.7). Abbildung A.22 ③ zeigt dies exemplarisch für das Feld Familienstand (*Maritalinformation.status*), dessen Wertebereich eingeschränkt ist (*:restrictedTo*) und mit einem Initialwert vorbelegt ist (*:initialValue*). Die **fachliche Semantik von Gruppen und Datenelementen** wird über eine *:semanticTag*-Annotation spezifiziert ④.

Das **Verhalten** wird über Annotationen zur Sichtbarkeitssteuerung (*:activeIf*, *:existsIf*) und für Operationen (*:actions*, *:reactions*, *:validations*) beschrieben[61]. In Abbildung A.22 ist dies für eine Existenzbedingung ⑤ und eine Reaktion ⑥ exemplarisch dargestellt.

A.5.2 Annotations-Profil zur Beschreibung der Korrelation zu Dienst-Ontologien

Die in Abschnitt 9.5 identifizierten Informationen zur Modellierung der semantischen und strukturellen Korrelation zwischen Elementen der Anwendungsontologie und der Dienst-Ontologie können analog den Basisinformationen des vorangegangenen Abschnitts als Element-Annotationen im RDF-Graphen angegeben werden.

Tabelle A.14 zeigt die Annotationen zur Beschreibung der semantischen und strukturellen Korrelation (Korrelations-Profil) und gibt an, auf welche Elemente der Applikations-Ontologie sie anwendbar sind.

Annotation	Anwendung auf	Beschreibung
Beschreibung der semantischen Korrelation		
:swClass	Gruppen `owl:Class, owl:ObjectProperties`	Klasse der korrellierenden Instanz in der Ziel-Ontologie
:swType	Datenelemente: `owl:DatatypeProperty`	Typ der zu erzeugenden Instanz in der Ziel-Ontologie
Beschreibung der strukturellen Korrelation		

:swIndividual	Gruppen und Datenelemente:
	`owl:Class,`
	`owl:ObjectProperties,`
	`owl:DatatypeProperty`
		Identität der zu erzeugenden Instanz in der Ziel-Ontologie.

:swForIndividual	Gruppen und Datenelemente:
	`owl:Class*,`
	`owl:ObjectProperties*,`
	`owl:DatatypeProperty`
		Identität der Instanz, der das Element zugeordnet wird

:swProperty	Gruppen und Datenelemente:
	`owl:Class*,`
	`owl:ObjectProperties*,`
	`owl:DatatypeProperty`
		Identität des Properties innerhalb der zugeordneten Instanz

* = lediglich notwendig, wenn Gruppe eine Untergruppe darstellt

Tabelle A.14. Annotationen des Korrelations-Profils

A.5.3 Herleitung einer Dienst-Ontologie-Instanz (DOI) aus einer Applikations-Ontologie-Instanz (AOI) zur Laufzeit

Aus der mit Korrelationsinformationen annotierten Applikations-Ontologie kann zur Laufzeit eine Dienst-Ontologie-Instanz (DOI) erzeugt werden. Hat der Nutzer die Instanzdaten erfasst, wird anhand der Informationen der damit beschriebene Graph für die Dienst-Ontologie-Instanz aufgebaut und mit den erfassten Instanzdaten versehen. Der folgende Algorithmus

[61] Zur Beschreibung des Verhaltens wird eine vereinfachte Notation verwendet, die auch im Rahmen der DSL genutzt wurde (s. Anhang A.3.1). Operationen werden in einem einzigen Attribut mit einer festgelegten Syntax beschrieben. Die explizite Modellierung des Verhaltens als eigenständige Entitäten der Ontologie ist jedoch möglich (vgl. Arbeiten von DeMeester et al.[138]), wird jedoch zukünftigen Arbeiten überlassen.

beschreibt das Verfahren zur Herleitung der DOI. Das Verfahren besteht in der Traversierung des in der Applikations-Ontologie beschriebenen Strukturbaums und der Auswertung der Instanzdaten, die für dessen Knoten erfasst wurden.

Algorithmus 8 Transformation der Instanzdaten auf eine Dienst-Ontologie-Instanz

```
1: function MAPAOITODOI(aoNode,AOI)
2:     inputs
3:         Strukturbaumknoten in Anwendungs-Ontologie (aoNode),
4:         Instanzdaten (AOI),
5:     result
6:         Ziel-Ontologie-Instanz (DOI)
7:     begin
8:         if aoNode ofType groupNode then
9:             for aoNode in aoNode.children do
10:                subtree ← mapAOItoDOI(aoNode)
11:                if subtree not empty then
12:                    containsData ← true;
13:            if containsData then
14:                DOI.addEntityInstance (aoNode.swIndividual, aoNode.swClass);
15:                if aoNode.swForIndividual then
16:                    DOI.addObjectPropertyInstance ( aoNode.swForIndividual, ao-
   Node.swProperty);
17:        if aoNode ofType dataelementNode then
18:            data ← AOI.getDataFor(aoNode);
19:            if data then
20:                DOI.addDatatypePropertyInstance (aoNode.swProperty, data, ao-
   Node.swType, aoNode.swForIndividual)
```

Ausgehend vom Wurzelknoten werden alle Knoten des Strukturbaums besucht.

- Handelt es sich bei einem besuchten Knoten um eine **Gruppe** mit annotierten Mapping-Informationen, wird anhand der Angaben eine korrespondierende *owl:Class*-Instanz erzeugt (Auswertung von *:swIndividual, :swClass*) und der Dienst-Ontologie-Instanz hinzugefügt. Ist die Beziehung zu einer anderen Entität angegeben (*:swForIndividual*), wird zusätzlich eine entsprechende *owl:ObjectProperty*-Instanz hinzugefügt, welche die Beziehung beschreibt.
- Handelt es sich bei dem besuchten Knoten um ein **Datenelement** und existieren hierfür Instanzdaten, wird anhand der Annotationen zum Typ (*:swType*) ein *owl:DatatypeProperty* erzeugt und der angegebenen Entität (*:swForIndividual*) als Property (*:swProperty*,) zugewiesen. Der Wert des *owl:DatatypeProperty* wird dabei mit den Instanzdaten belegt.

Das Resultat ist eine Ontologie, die aus Elementen der Dienst-Ontologie besteht und die Struktur aufweist, die in der Applikations-Ontologie angegeben wurde.

A.6 Beispielergebnisse der Evaluationsphase

Dieser Abschnitt stellt beispielhaft Ergebnisse dar, die im Rahmen der Evaluation entstanden sind. Zudem werden Verweise auf zusätzliche Materialen angegeben. Sie dienen der Illustration der Einsatzmöglichkeiten des vorgestellten Ansatzes in unterschiedlichen Szenarien und der Qualität der generierten Benutzungsschnittstellen, die bei der Umsetzung erreicht wurde. Hierzu werden im Folgenden Bildschirmfotos und Verweise auf lauffähige Anwendungen bzw. Videos angegeben. Unter

- ***http://thesis.mimesis.solutions/index.evaluation-material.html***

sind die Verweise zusammengestellt, sofern es sich zum Zeitpunkt der Erstellung der Arbeit um frei verfügbare Ressourcen handelte.

A.6.1 Produktiver Einsatz in *Anwendungen zur Risikoanalyse*

In Abschnitt 10.4.1 wurde die *Action Research* Studie zur automatischen Generierung von Benutzungsschnittstellenvarianten dargestellt. Dort wurden Dialoge zur Risikobewertung bereits mit Abbildungen illustriert. Die folgenden Abbildungen zeigen weitere, automatisch generierte Ergebnisse für Fragen zur Risikobewertung. Diese werden in mehreren Antragstrecken zum Abschluss von Versicherungsprodukten in Varianten verwendet. Die Risikofragen werden sowohl in Vermittleranwendungen im Vermittlerportal als auch in Endkunden-Anwendungen im Internetauftritt der Allianz verwendet (vgl. Abschnitt 10.4.1).

Für die Varianten der Risikofragen wurden abstrakte Benutzungsschnittstellenbeschreibungen erzeugt und daraus erforderliche technische Varianten zur Laufzeit generiert. Die folgenden Abbildungen wurden für Antragstrecken im Kundenportal erzeugt. Während im Vertreterportal Standard-Interaktionselemente zur Eingabe verwendet wurden, bestand hier der Bedarf an optimierten Interaktionselementen, welche die Eingabe der geforderten Daten für Endkunden vereinfachen sollen. Die Darstellungen wurden den Antragstrecken für eine *Risiko-Lebensversicherung* bzw. *Berufsunfähigkeits-Versicherung* entnommen, die zum Zeitpunkt der Erstellung der Arbeit unter folgenden Verweisen erreichbar sind.

- *https://www.allianz.de/vorsorge/rechner/risikoleben/antrag/*
- *https://www.allianz.de/vorsorge/berufsunfaehigkeitsversicherung/antrag/*

Einbettung der Risikofragen: Abbildung A.23 zeigt die Seite der Antragstrecke zu einer Risikolebensversicherung, die Fragedialoge für Risikofragen integriert. Die Risikofragen sind dabei als Teil einer Rahmenanwendung in deren Anwendungsfluss integriert. Im oberen Bereich werden Fragen zur beruflichen Situation eingebettet. Im unteren Bereich befinden sich Fragen zur Konstitution, Gesundheit und Auslandsaufenthalten. Die Navigation erfolgt hier über *Weiter*-Schaltflächen, die bei angeschlossener Eingabe auf die nächste Frageseite führen bzw. den Fragedialog abbrechen.

Angabe der beruflichen Situation: Abbildung A.24 zeigt die Fragen zur beruflichen Situation. Hierfür wurde eine 1zu-N-Auswahl verwendet, die als anwendungsspezifisches Interaktionselement dargestellt wird. Dieses wurde mit einer semantischen Annotation versehen,

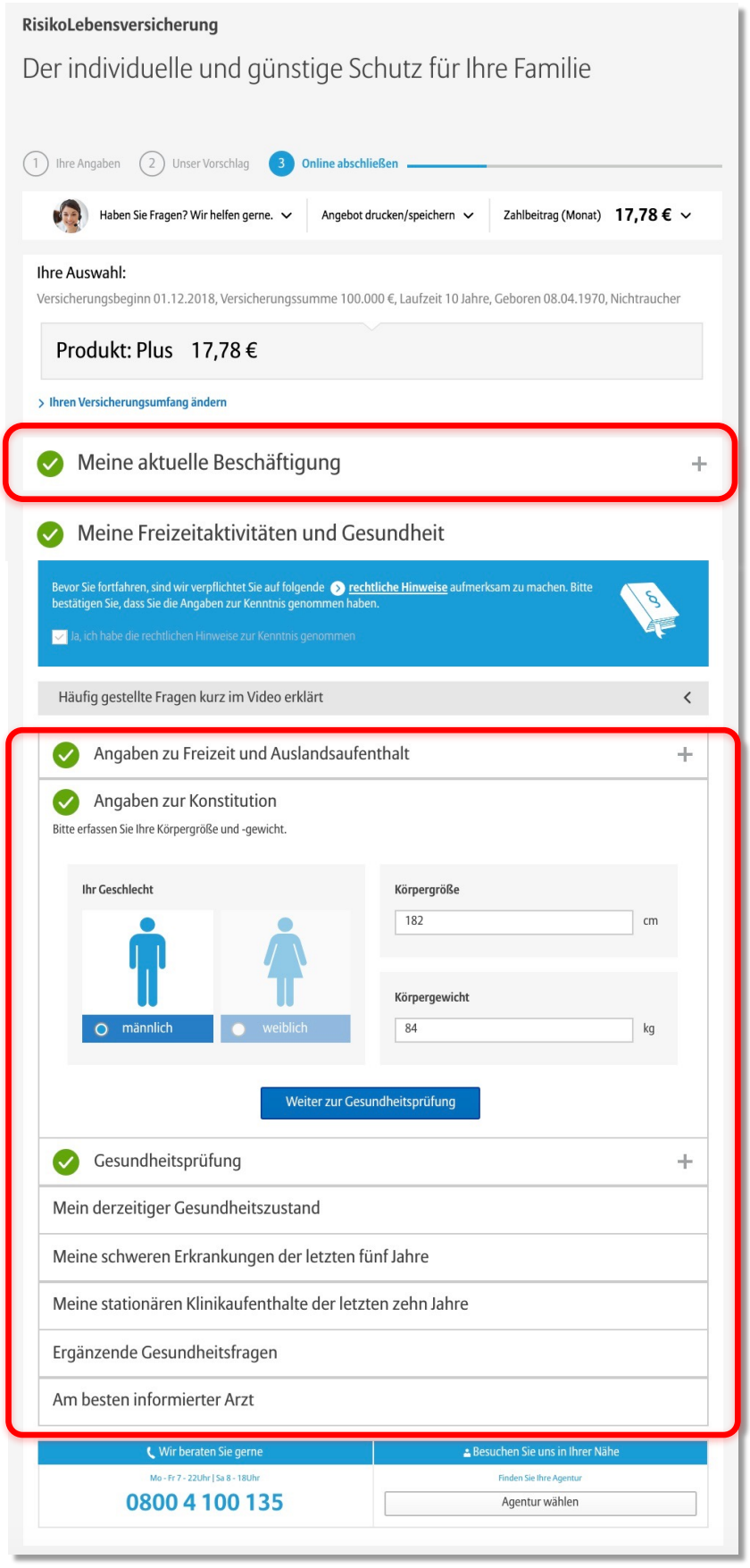


Abbildung A.23. Einbettung der Riskofragen in die Antragstrecke

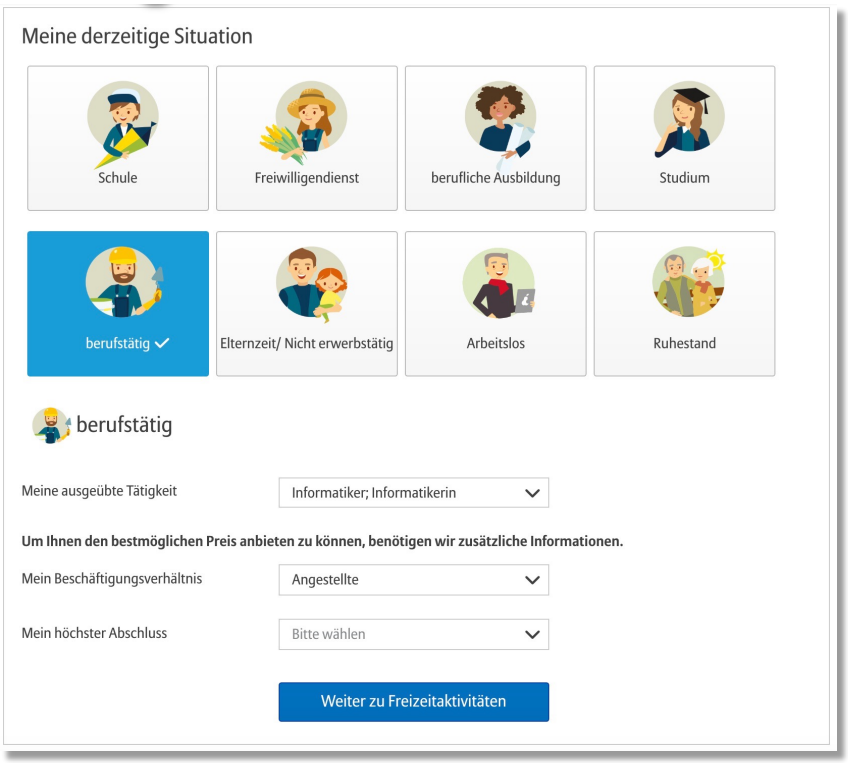


Abbildung A.24. Berufliche Situation

welche zur dargestellten Variante führt. Bei Auswahl eines Icons zur beruflichen Situation werden im unteren Bereich der Dialogs weitere Nachfragen gestellt.

Angabe ausgeübter Sportarten: Abbildung A.25 zeigt eine ähnliche Darstellung für die die Fragen zu ausgeübten Sportarten. Hier entspricht die Auswahl jedoch einer M-aus-N-Auswahl, die eine Mehrfachselektion zulässt. Dementsprechend werden weitere Fragegruppen folgen, die zu den ausgewählten Kategorien passen.

Gerätespezifische Alternativen: Abbildung A.26 zeigt zwei alternative Interaktionselemente, die zur Eingabe der Konstitution verwendet wurden. Die erste Alternative (links) wird auf Geräten ohne Größenrestriktionen verwendet. Für größenbeschränkte Geräte erfolgt eine kompaktere Darstellung (rechts).

Angabe zu Erkrankungen über einen *Körperatlas*: Abbildung A.27 zeigt die Erfassung erkrankter Körperbereiche in Form eines spezifischen anwendungsspezifischen Interaktionselemens (Körperatlas). Die Eingabe ist eine M-aus-N-Auswahl, die mit der semantischen Information versehen wurde, dass es sich um Fragen zur Körperregion handelt und ein kreisförmiges Eingabeelement gewählt werden soll.

Nach Auswahl eines Körperbereichs erfolgt im unteren Bereich die Auswahl von Erkrankungen. Zu den dort angegebenen Erkrankungen werden weiterführende Fragen gestellt. In Abbildung A.27 (unten) erfolgte die Auswahl einer *Refluxösophagitis*, zu der Fragen zum Verlauf der Erkrankung gestellt werden.

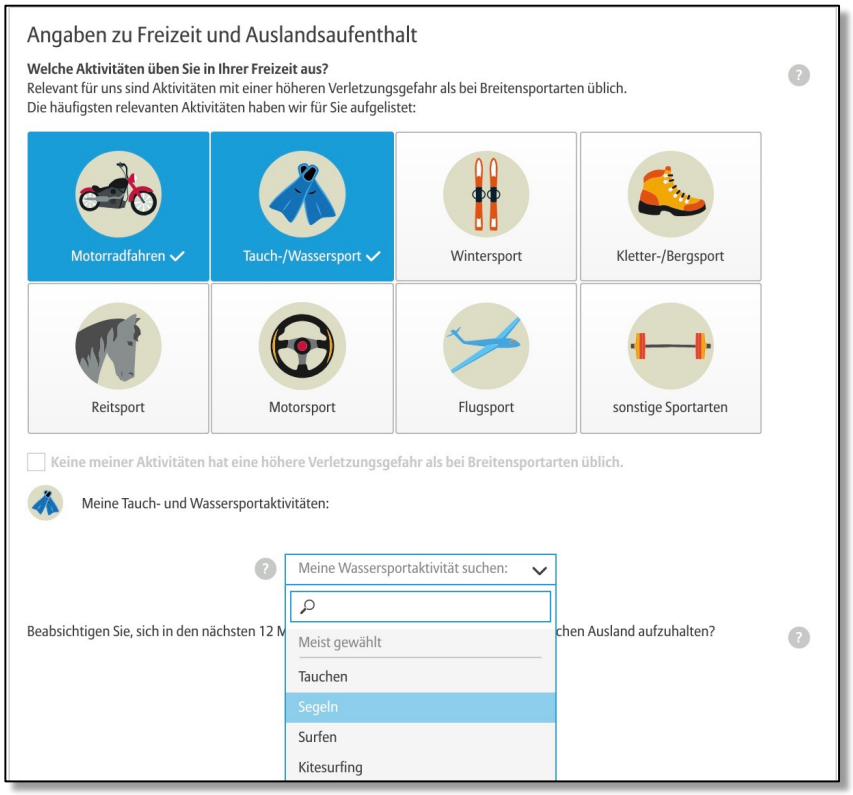


Abbildung A.25. Ausgeübte Sportarten

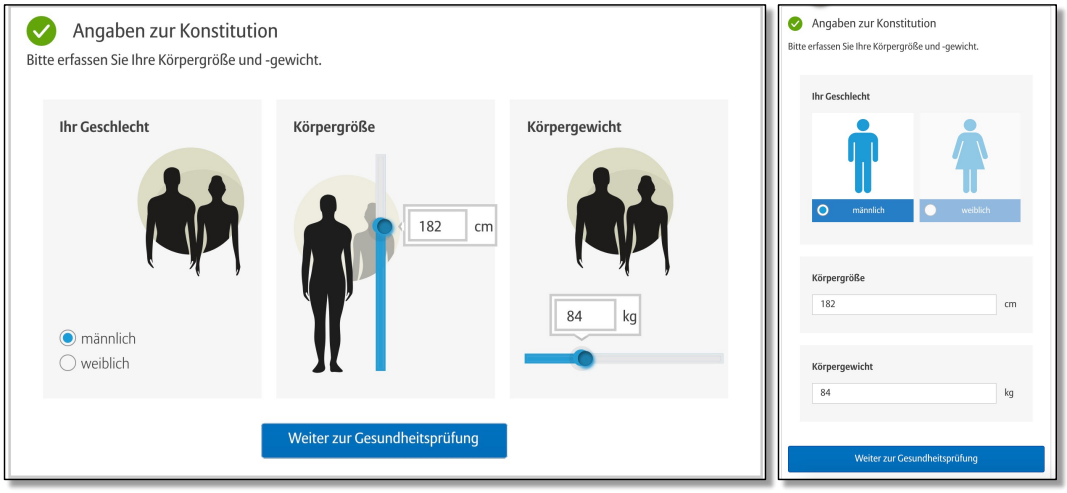


Abbildung A.26. Gerätespezifische Varianten (Konstitution)

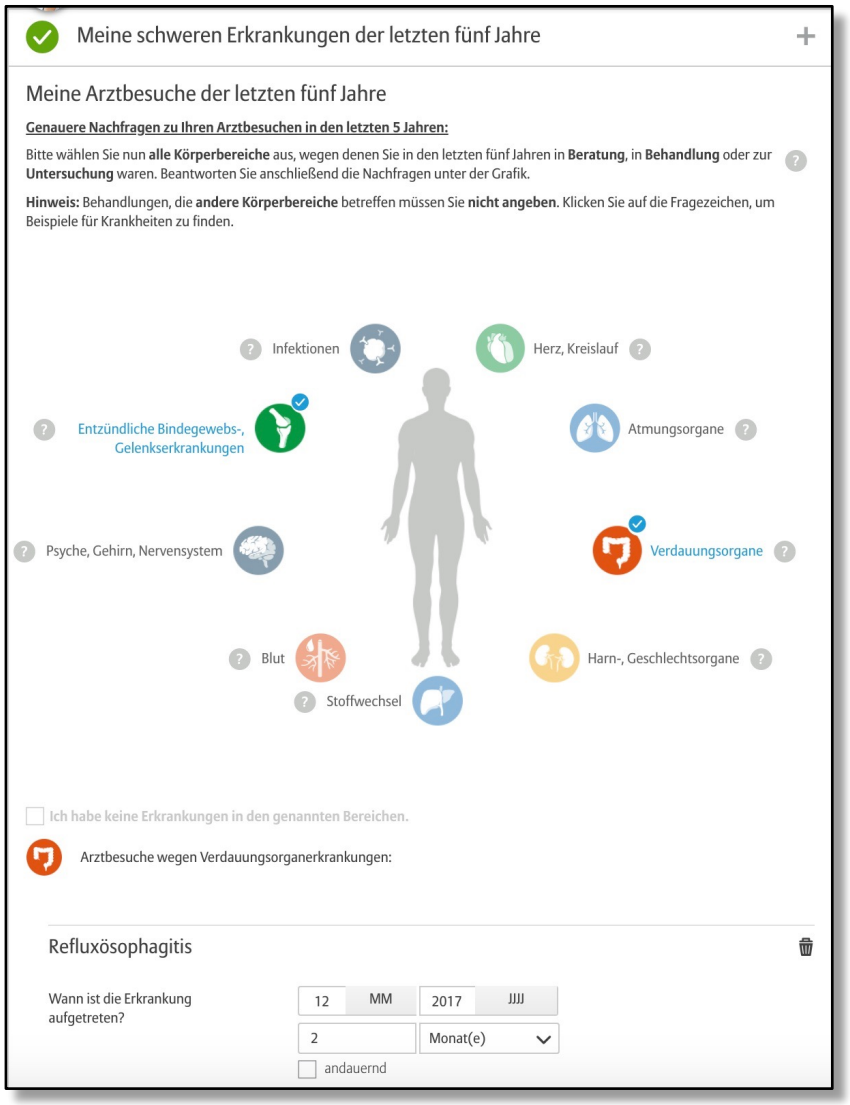


Abbildung A.27. Erfassung von Erkrankungen über Körperatlas

A.6.2 Freie Kombination von Dienstleistungen in einem *Dienstleistungs-Selektor*

In Abschnitt 10.4.2 wurde die Fallstudie zur freien Kombination von Diensten in einem Service Selektor dargestellt und bereits mit Abbildungen illustriert.

Da es sich bei diesem Demonstrator um eine unabhängige Fallstudie handelt, ist dieser frei verfügbar und kann begutachtet werden. Unter der in der Einleitung genannten Evaluations-Webseite befindet sich sowohl ein Verweis auf die Anwendung als auch ein Video, welches die Funktionsweise demonstriert.

- URL: *http://thesis.mimesis.solutions/index.evaluation-material.html*
- Bereich : *Case Study: Service-Selector*

A.6.3 Shareable UIs für Distributed Marketspaces

In Abschnitt 10.4.3 wurde die *Action Research* Studie zur Erfassung der Daten für einen *Complex Product Request* dargestellt und bereits mit Abbildungen illustriert.

Da es sich hierbei um ein nicht öffentlich zugängliches Projekt handelt, kann die laufende Anwendung nicht zur Verfügung gestellt werden. Unter der in der Einleitung genannten Evaluations-Webseite befindet sich jedoch der Verweis auf ein Video, welches die Funktionsweise demonstriert. Zudem sind hier zur Untersuchung die Anwendungsmodelle referenziert, die als Komponenten zur Erzeugung der Benutzungsschnittstelle kombiniert werden konnten.

- URL: *http://thesis.mimesis.solutions/index.evaluation-material.html*
- Bereich : *Action Research: Distributed Makrtspaces*

A.7 Online verfügbare Ergebnisse der Arbeit

Um die im Rahmen der Arbeit verwendeten Beispiele vollständig inspizieren und im Zuge der Validierung verwendeten Anwendungen interaktiv untersuchen zu können, wurde eine Projekt-Website angelegt, die unter folgender URL erreichbar ist:

- ***http://thesis.mimesis.solutions***

Die Website ist in folgende Themenbereiche gegliedert, in denen zusätzliche Materialien angeboten werden:

- Materialien der Arbeit
- Materialien zur Validierung
- Materialien zur Evaluation
- Materialien zur Implementierung

A.7.1 Materialen der Arbeit

Dieser Bereich enthält die vollständigen Artefakte der in den Kapiteln der Arbeit verwendeten Beispiele.

- *http://thesis.mimesis.solutions/index.thesis-material.html*

A.7.2 Materialien zur Validierung

Der Bereich enthält verweise auf lauffähige Versionen und vollständige Artefakte ausgewählter Anwendungen, die u.a. im Zuge der Validierung genutzt wurden und den Funktionsumfang der prototypischen Implementierung zeigen.

- *http://thesis.mimesis.solutions/index.validation-material.html*

A.7.3 Materialien zur Evaluation

Der Bereich enthält Verweise auf verfügbare Anwendungen, Videos und Materialien der Evaluationsphase.

- *http://thesis.mimesis.solutions/index.evaluation-material.html*

A.7.4 Materialien zur Implementierung

Dieser Abschnitt enthält Verweise auf die Online-Dokumentation der Schnittstellen (APIs) der im Rahmen der prototypischen Implementierung umgesetzten Infrastruktur-Dienste. Zudem kann hier die Version einer Rich-Client-Anwendung heruntergeladen werden, welche die Umsetzung der Benutzerschnittstellenbeschreibungen für Desktop-Anwendungen demonstriert.

- *http://thesis.mimesis.solutions/index.implementation-material.html*

Zur Umsetzung eines Benutzungsschnittstellen-Ökosystems aus Kapitel 9 sind als zentrale Infrastrukturkomponenten *UI-Dienste*, *Anwendungsmodell-Repositories* und *Instanzdaten-Dienste* vorgesehen. Im Rahmen der Evaluierungen wurden diese prototypisch als *RESTful*-Webservices umgesetzt (s. Abschnitt 10.2.2).

Unter der angegebenen URL kann auf die Dokumentation der implementierten APIs zugegriffen werden. Die einzelnen Operationen der APIs sind dort beschrieben und können über eine GUI getestet werden. Abbildung A.28 zeigt den Einstiegspunkt der Dokumentation. Rechts oben kann der Infrastruktur-Dienst ausgewählt und dann im unteren Bereich die Dokumentation inspiziert werden. Abbildung A.29 zeigt exemplarisch die Dokumentation einer Operation des *mimeisis.ui.service* (hier die Generierung einer finalen UI aus *Shared UIs*). Mittels der Schaltfläche *Try it out* können die Eingabeparameter angegeben und die Operation ausgeführt werden.

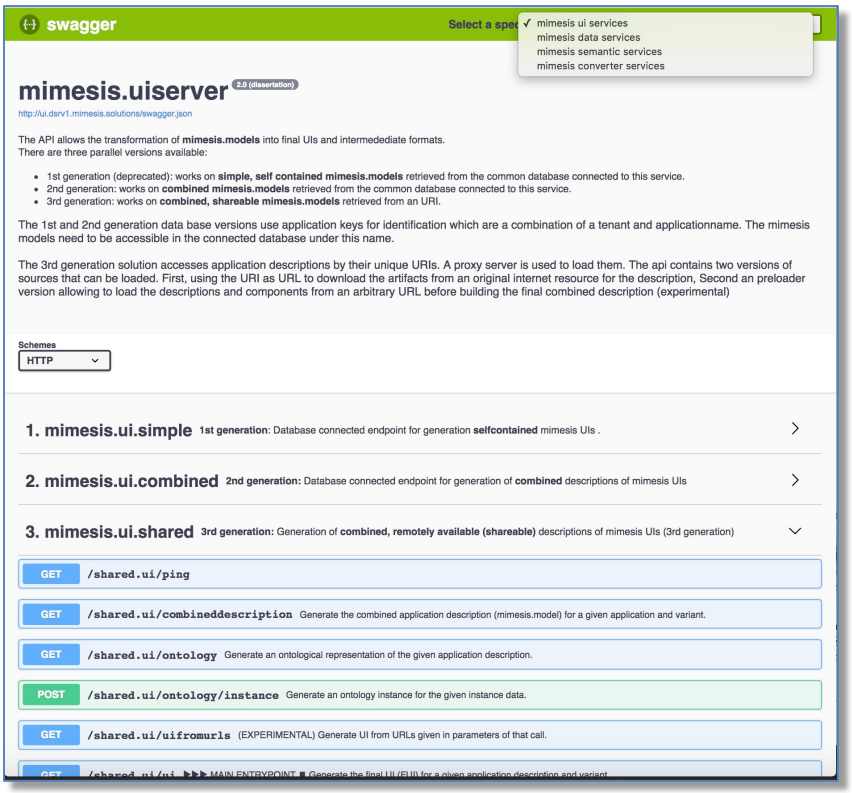


Abbildung A.28. Dokumentation der Infrastrukturkomponenten-APIs

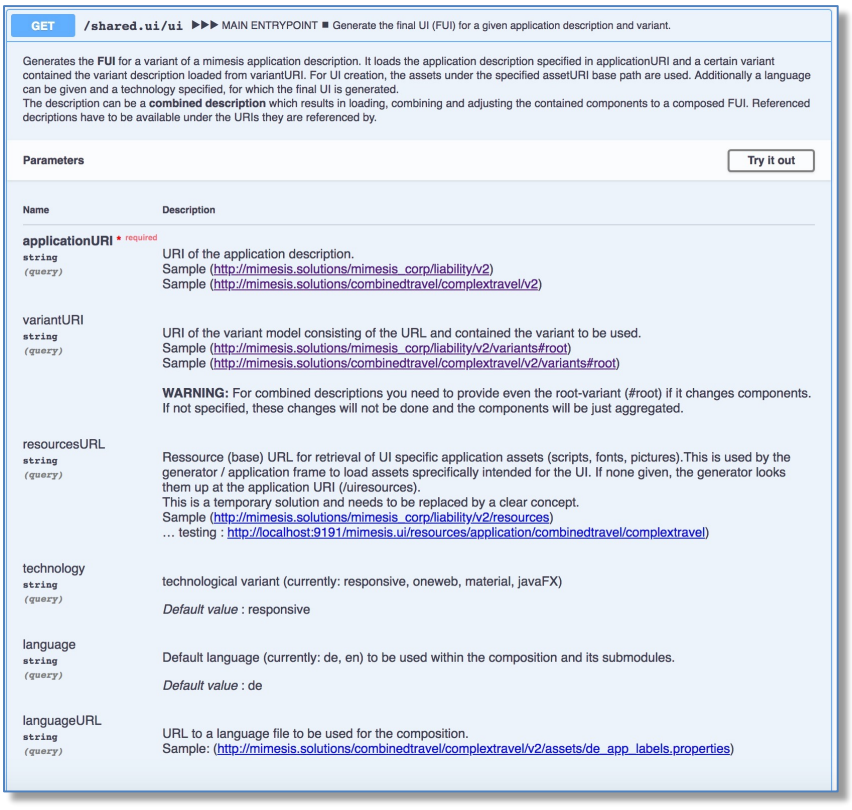


Abbildung A.29. Dokumentation einer API-Operation

A.8 Source-Listings

A.8.1 mimesis.model.DSL: Antragstrecke Haftpflichtversicherung

```
data_description : "liability"
   referencedOntologies=[ext:http://mimesis.solutions/types/ext/v1#]
   uri="http://mimesis.solutions/mimesis_corp/liability/v2#"
   _version="2.0.0"

{
    group : "introduction"
    {
       ...
       loaddata : {
           type="actionbutton"
           actions="element:click:fn_liab.loadModel()"
       }
    }
    group : "customerdata"
    {
        group : "fullname"  {
              actions="button:click:appactions.openCustomerDB($liability.customerdata)"
        }
        {
           gender : {
               restrictedTo="male|female"
           }
           firstname : {
               required
           }
           secondname : {
               activeIf="liability.customerdata.fullname.firstname.length>0"
               initialValue=""
           }
           lastname : {
               actions="info:click:appactions.showInfo($liability.customerdata.fullname....
                    ↪ lastname)"
               required
           }
        }
        group : "birthdata"
        {
           dateofbirth : {
               type="date"
               initialValue="1970-12-11T00:00:00.000"
               required
           }
           placeofbirth : {
           }
        }
        group : "maritalinformation"
        {
           status : {
               restrictedTo="married|notmarried|divorced"
               actions="info:click:appactions.showInfo($qqq)"
               initialValue="married"
               required
           }
           marriedsince : {
               existsIf="(liability.customerdata.maritalinformation.status == 'married')...
                    ↪ "
               type="date"
               actions="info:click:appactions.showInfo($liability.customerdata....
                    ↪ maritalinformation.marriedsince)"
           }
        }
```

```
60          group : "partnerinformation"  {
61                existsIf="(liability.customerdata.maritalinformation.status == 'married')"
62                cohesion="weak"
63          }
64          {
65             spouse_firstname : {
66                 required
67             }
68             spouse_lastname : {
69                 required
70             }
71          }
72          group : "children"
73          {
74             numberOfChildren : {
75                 min="0"
76                 validations="validation.checkChildrenAmount(liability.customerdata....
                       ↪ children.numberOfChildren)"
77                 type="number"
78                 max="11"
79             }
80             childrenbelow7 : {
81                 existsIf="(liability.customerdata.children.numberOfChildren > 0)"
82                 validations="validation.checkChildren(liability.customerdata.children....
                       ↪ numberOfChildren,liability.customerdata.children.childrenbelow7)"
83                 type="number"
84             }
85          }
86          group : "address"  {
87                cohesion="weak"
88          }
89          {
90             street : {
91             }
92             building_no : {
93                 semanticTags="ext:buildingno"
94             }
95             zip : {
96                 type="zip"
97                 semanticTags="ext:zipcode"
98             }
99             city : {
100                reactions="liability.customerdata.address.zip:fn_liab.prefillCity(...
                       ↪ liability.customerdata.address.zip,$liability.customerdata.address...
                       ↪ .city)"
101            }
102            country : {
103                type="country"
104                restrictedTo="fn_liab.getCountries()"
105            }
106            eMail : {
107                semanticTags="ext:email"
108                required
109            }
110            phone : {
111                semanticTags="ext:phone"
112            }
113         }
114      }
115      group : "contractdata"
116      {
117         kindofliability : {
118             character="distinctSelect"
119             restrictedTo="family|couple|single"
120             initialValue="single"
121         }
122          group : "partnertoinclude"  {
123                existsIf="(liability.contractdata.kindofliability != 'single')"
```

```
124                cohesion="weak"
125            }
126            {
127               gender : {
128                   restrictedTo="female|male"
129               }
130               firstname : {
131                   required
132               }
133               secondname : {
134               }
135               lastname : {
136                   required
137               }
138               dateofbirth : {
139                   type="date"
140                   initialValue="1980-3-3T00:00:00.000Z"
141               }
142               placeofbirth : {
143               }
144            }
145            group : "rabate_yp"
146            {
147               youngpeoplerabate : {
148                   activeIf="dateAfter(liability.customerdata.birthdata.dateofbirth, ...
                          ↪ '1990-01-01T00:00:00.000Z')"
149                   type="boolean"
150               }
151            }
152            group : "previousdamages"
153            {
154               haspreviousdamages : {
155                   type="boolean"
156                   initialValue="false"
157               }
158               numberofdamages : {
159                   type="number"
160                   existsIf="(liability.contractdata.previousdamages.haspreviousdamages ==...
                          ↪ true)"
161               }
162               damagedescription : {
163                   type="longtext"
164                   existsIf="(liability.contractdata.previousdamages.haspreviousdamages ==...
                          ↪ true)"
165               }
166            }
167            group : "productconfiguration"  {
168                  cohesion="weak"
169            }
170            {
171               basecomponent : {
172                   character="distinctSelect"
173                   restrictedTo="extendedplusliability|extendedliablitiy|basicliability"
174                   initialValue="extendedliablitiy"
175               }
176                group : "component_safety"  {
177                      existsIf="(liability.contractdata.productconfiguration.basecomponent ...
                             ↪ == 'extendedplusliability')"
178                }
179                {
180                   safetybest : {
181                       activeIf="(liability.contractdata.productconfiguration.basecomponent ...
                              ↪ == 'extendedplusliability')"
182                       type="boolean"
183                       initialValue="false"
184                   }
185                }
186                group : "component_oiltank"  {
```

```
187                 existsIf="(liability.contractdata.productconfiguration.basecomponent ...
                        ↪ != 'basicliability')"
188             }
189             {
190                oiltankplus : {
191                    type="boolean"
192                    initialValue="false"
193                }
194                buildingusedbyIP : {
195                    existsIf="(liability.contractdata.productconfiguration....
                           ↪ component_oiltank.oiltankplus==true)"
196                    type="boolean"
197                    initialValue="false"
198                }
199             }
200             group : "tanklocation"  {
201                   existsIf="(liability.contractdata.productconfiguration....
                           ↪ component_oiltank.oiltankplus==true) && (liability.contractdata...
                           ↪ .productconfiguration.component_oiltank.buildingusedbyIP==true)...
                           ↪ "
202                   cohesion="weak"
203             }
204             {
205                street : {
206                }
207                building_no : {
208                    semanticTags="ext:buildingno"
209                }
210                zip : {
211                    semanticTags="ext:zipcode"
212                }
213                city : {
214                }
215                country : {
216                    type="country"
217                    restrictedTo="fn_liab.getCountries()"
218                }
219                locationchecked : {
220                    type="boolean"
221                    initialValue="false"
222                }
223             }
224             group : "component_landlord"  {
225                   existsIf="(liability.contractdata.productconfiguration.basecomponent ...
                           ↪ != 'basicliability')"
226             }
227             {
228                landlordplus : {
229                    type="boolean"
230                    initialValue="false"
231                }
232                numberOfFlats : {
233                    existsIf="(liability.contractdata.productconfiguration....
                           ↪ component_landlord.landlordplus==true)"
234                    min="0"
235                    type="number"
236                    max="3"
237                    initialValue="0"
238                }
239             }
240             group : "flatlocations"  {
241                   existsIf="(liability.contractdata.productconfiguration....
                           ↪ component_landlord.landlordplus==true) && (liability....
                           ↪ contractdata.productconfiguration.component_landlord....
                           ↪ numberOfFlats>0)"
242                   cohesion="weak"
243             }
244             {
```

```
245                 group : "landlordflat_1"  {
246                       existsIf="(liability.contractdata.productconfiguration....
                              ↪ component_landlord.numberOfFlats > 0)"
247                 }
248                 {
249                    street : {
250                    }
251                    building_no : {
252                        semanticTags="ext:buildingno"
253                    }
254                    zip : {
255                        semanticTags="ext:zipcode"
256                    }
257                    city : {
258                    }
259                    country : {
260                        type="country"
261                        restrictedTo="fn_liab.getCountries()"
262                    }
263                 }
264                 group : "landlordflat_2"  {
265                       existsIf="(liability.contractdata.productconfiguration....
                              ↪ component_landlord.numberOfFlats > 1)"
266                 }
267                 {
268                    street : {
269                    }
270                    building_no : {
271                        semanticTags="ext:buildingno"
272                    }
273                    zip : {
274                        semanticTags="ext:zipcode"
275                    }
276                    city : {
277                    }
278                    country : {
279                        type="country"
280                        restrictedTo="fn_liab.getCountries()"
281                    }
282                 }
283                 group : "landlordflat_3"  {
284                       existsIf="(liability.contractdata.productconfiguration....
                              ↪ component_landlord.numberOfFlats >2)"
285                 }
286                 {
287                    street : {
288                    }
289                    building_no : {
290                        semanticTags="ext:buildingno"
291                    }
292                    zip : {
293                        semanticTags="ext:zipcode"
294                    }
295                    city : {
296                    }
297                    country : {
298                        type="country"
299                        restrictedTo="fn_liab.getCountries()"
300                    }
301                 }
302              }
303           }
304           group : "payment"  {
305                 cohesion="weak"
306           }
307           {
308              ownrisk : {
309                  restrictedTo="none|150|250|500|1000|2500"
```

```
310             }
311             paymentperiod : {
312                 restrictedTo="month|quarter|halfyear|year"
313             }
314             duration : {
315                 restrictedTo="oneyear|threeyears"
316             }
317         }
318         group : "agreements"  {
319               cohesion="weak"
320         }
321         {
322             rabatesapply : {
323                 type="boolean"
324                 initialValue="false"
325             }
326              group : "bundlerabate"  {
327                    existsIf="(liability.contractdata.agreements.rabatesapply == true)"
328              }
329              {
330                 applybundlerabate : {
331                     activeIf="(liability.contractdata.agreements.rabatesapply == true) &&...
                             ↪  (liability.contractdata.agreements.specialcases....
                             ↪ applyspecialcase == false)"
332                     type="boolean"
333                     initialValue="false"
334                 }
335              }
336              group : "goodcustomerrabate"  {
337                    existsIf="(liability.contractdata.agreements.rabatesapply == true)"
338              }
339              {
340                 applygoodcustomerrabate : {
341                     activeIf="(liability.contractdata.agreements.specialcases....
                             ↪ applyspecialcase == false)"
342                     type="boolean"
343                     initialValue="false"
344                 }
345                 rabate_gc : {
346                     semanticTags="ext:percentage"
347                     existsIf="(liability.contractdata.agreements.goodcustomerrabate....
                             ↪ applygoodcustomerrabate == true)"
348                 }
349              }
350              group : "specialcases"  {
351                    existsIf="(liability.contractdata.agreements.rabatesapply == true)"
352              }
353              {
354                 applyspecialcase : {
355                     activeIf="(liability.contractdata.agreements.goodcustomerrabate....
                             ↪ applygoodcustomerrabate == false) && (liability.contractdata....
                             ↪ agreements.bundlerabate.applybundlerabate == false)"
356                     type="boolean"
357                     initialValue="false"
358                 }
359                 cases : {
360                     restrictedTo="none|frameworkagreement_before_0108|...
                             ↪ frameworkagreement_after_0108|procuration"
361                     existsIf="(liability.contractdata.agreements.specialcases....
                             ↪ applyspecialcase == true)"
362                     initialValue="none"
363                 }
364                 rabate_sc : {
365                     semanticTags="ext:percentage"
366                     existsIf="(liability.contractdata.agreements.specialcases.cases != '...
                             ↪ none') && (liability.contractdata.agreements.specialcases....
                             ↪ applyspecialcase == true)"
367                 }
```

```
                statement : {
                    type="longtext"
                    existsIf="(liability.contractdata.agreements.specialcases.cases != '...
                        ↪ none') && (liability.contractdata.agreements.specialcases....
                        ↪ cases != 'procuration') && (liability.contractdata.agreements....
                        ↪ specialcases.applyspecialcase == true)"
                }
            }
        }
    }
    group : "result"
    {
       summary : {
           type="snippet"
           initialValue="snippets/summary.html"
       }
       savedata : {
           type="actionbutton"
           actions="element:click:fn_liab.saveModel()"
       }
    }
    group : "inspect"
    { ...    }
}
```

Listing A.1: Anwendungsmodell Haftpflichtversicherung

A.8.2 mimesis.variant.DSL: Antragstrecke Haftpflichtversicherung

```
variant_description : {
    variant : "root" {
        uri : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#root"
    }
    variant : "agent_desktop" {
        uri : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#agent_desktop"
        extends : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#root"
    }
    variant : "customer_internet" {
        uri : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#...
            ↪ customer_internet"
        extends : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#root"

        exclude : "liability.customerdata.fullname.secondname"
        exclude : "liability.customerdata.partnerinformation"
        exclude : "liability.contractdata.rabate_yp"
        exclude : "liability.contractdata.previousdamages",
        include : "liability.contractdata.previousdamages.haspreviousdamages"
        include : "liability.contractdata.previousdamages.numberofdamages"
        exclude : "liability.contractdata.productconfiguration.component_oiltank"
        include : "liability.contractdata.productconfiguration.component_oiltank....
            ↪ oiltankplus"
        exclude : "liability.contractdata.productconfiguration.tanklocation"
        exclude : "liability.contractdata.productconfiguration.component_landlord"
        include : "liability.contractdata.productconfiguration.component_landlord....
            ↪ landlordplus"
        include : "liability.contractdata.productconfiguration.component_landlord....
            ↪ numberOfFlats"
        exclude : "liability.contractdata.productconfiguration.flatlocations"
        exclude : "liability.contractdata.agreements"
        exclude : "liability.data"


        modify : {
```

```
31              type        : "attribute"
32              element       : "liability.customerdata.address.city"
33              attribute    : "initialValue"
34              newvalue    : "Stuttgart"
35          }
36
37          modify : {
38              type        : "attribute"
39              element       : "liability.customerdata.fullname.firstname"
40              attribute    : "initialValue"
41              newvalue    : "Mobile_Max"
42          }
43
44          modify : {
45              type                : "operation"
46              element               : "liability.customerdata.fullname"
47              modification        : "remove"
48              operationType        : "action"
49              operationID           : "appactions.openCustomerDB"
50          }
51      }
52      variant: "customer_mobile" {
53
54          uri : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#...
                ↪ customer_mobile"
55          extends : "http://mimesis.solutions/mimesis_corp/liability/v2/variants#...
                ↪ customer_internet"
56
57          exclude : "liability.introduction",
58          exclude : "liability.customerdata.birthdata",
59          exclude : "liability.customerdata.maritalinformation.marriedsince",
60          exclude : "liability.customerdata.children.childrenbelow7",
61          exclude : "liability.customerdata.address.street",
62          exclude : "liability.customerdata.address.zip",
63          exclude : "liability.customerdata.address.city",
64          exclude : "liability.customerdata.address.country",
65          exclude : "liability.customerdata.address.building_no",
66
67          exclude : "liability.contractdata.rabate_yp",
68          exclude : "liability.contractdata.partnertoinclude.secondname",
69          exclude : "liability.contractdata.partnertoinclude.dateofbirth",
70          exclude : "liability.contractdata.partnertoinclude.placeofbirth",
71          exclude : "liability.contractdata.productconfiguration.component_landlord",
72          exclude : "liability.contractdata.productconfiguration.component_oiltank",
73          exclude : "liability.contractdata.agreements"
74
75      }
76  }
```

Listing A.2: Variantenbeschreibung Haftpflichtversicherung

A.8.3 mimesis.model.DSL: Kombinierte Reisebuchung

Das folgende Listing zeigt ein leicht vereinfachtes Anwendungsmodell für eine umfangreiche Reisebuchung. Es handelt sich um eine Rahmenanwendung, in welcher Grunddaten und optionale Komponenten einer Reise erfasst werden. Die Komponenten werden in eigenständigen Modellen beschrieben und zur Gesamtanwendung zusammengesetzt. Ein Beispiel ist die Flugbuchungs-Komponente (vgl. Anhang A.8.4). In Anhang A.8.5 wird das zugehörige Variantenmodell dargestellt, welches die sehr unterschiedlichen Komponenten gemäß der geänderten Nutzung modifiziert und nahtlos zu einer neuen Anwendung zusammenfügt.

```
data_description : "tripbooking"
uri="http://bookers.mimesis.solutions/tripbooking/mw2020"
_version = "2.0.43"
referencedOntologies = [
    base: http://bookers.mimesis.solutions/tripbooking/mw2020#,
    ext: http://mimesis.solutions/types/ext/v1#,
    bnk: http://mimesis.solutions/types/banking/v1#,
    tiod: http://products.org/ticketing/v1#,
    gr : http://purl.org/goodrelations/v1#,
    cmh : http://buxxi.org/carrymehome/v1#,
    fbo : http://mimesis.soultions/flightbooking/v1#,
    foaf: http://schema.org/foaf/v1#,
    xmls: http://www.w3.org/2001/XMLSchema#
  ]
{
  group: "basicdata"
  {
    group: "travellers"
    {
      gender : {
        type="oneOfMany"
        restrictedTo="male|female" initialValue="male"
      }
      firstname : {}
      lastname  : {}
       group : "persons"
       {
        adults   : {type="number" initialValue="1" min="1" max="3"}
        children : {type="number" min="0" max="3" initialValue="2"}
      }
    }
    group: "traveldates"
    {
      startdate : {type="date" initialValue="mimesis.date.today()"}
      enddate   : {type="date" initialValue="mimesis.date.today()"}
    }
    group: "destinations"
    {
      start :{
        character="iconicSelect"
        restrictedTo="fn_flbook.getDepartureAirports()"
      }
      destination :{
        character="distinctSelect" restrictedTo="selectstart"
        cardinality="1"
        type="text"
        reactions="tripbooking.basicdata.destinations.start:fn_flbook.changeDestinations(...
            ↪ tripbooking.basicdata.destinations.start,$tripbooking.basicdata....
            ↪ destinations.destination)"
        existsIf="tripbooking.basicdata.destinations.start.length>0"
      }
    }
  }
  group : "options"
  {
    components : { type="manyOfMany",
             character="iconicSelectMany",
             restrictedTo="flight|car|overnightstay|babysitter|concert"
             initialValue="" }

    group: "additional"
    {
      travelinsurance : { type = "boolean" initialValue= "false"}
    }

    //
    // Referenced Components
    //
```

```
67
68      component : "flightbooking"
69      {
70        uri ="http://aviation.mimesis.solutions/products/flightbooking/mw2020"
71        variant ="http://aviation.mimesis.solutions/products/flightbooking/mw2020/variants#...
              ↪ root"
72        cohesion ="weak"
73        existsIf ="contains(tripbooking.options.components,'flight')"
74      }
75
76      component : "carrental"
77      {
78        uri ="http://bookers.mimesis.solutions/products/carrental/mw2020"
79        variant ="http://bookers.mimesis.solutions/products/carrental/mw2020/variants#root"
80        cohesion ="weak"
81        existsIf ="contains(tripbooking.options.components,'car')"
82      }
83
84      component : "overnightstay"
85      {
86        uri ="http://bookers.mimesis.solutions/products/overnightstay/v1"
87        variant ="http://bookers.mimesis.solutions/products/overnightstay/v1/variants#root"
88        cohesion="weak"
89        existsIf="contains(tripbooking.options.components,'overnightstay')"
90      }
91
92      component : "babysitter"
93      {
94        uri   ="http://bookers.mimesis.solutions/products/babysitting/v1"
95        variant ="http://bookers.mimesis.solutions/products/babysitting/v1/variants#root"
96        cohesion="weak"
97        existsIf="contains(tripbooking.options.components,'babysitter')"
98      }
99
100     component : "concert"
101     {
102       uri   ="http://bookers.mimesis.solutions/products/concert/v1"
103       variant ="http://bookers.mimesis.solutions/products/concert/v1/variants#root"
104       cohesion="weak"
105       existsIf="contains(tripbooking.options.components,'concert')"
106     }
107   }
108
109   component : "insurance"
110   {
111     uri   ="http://insurance.mimesis.solutions/travelinsurance/v1"
112     variant ="http://insurance.mimesis.solutions/travelinsurance/v1/variants#root"
113     cohesion="weak"
114     existsIf="(tripbooking.options.additional.travelinsurance == true)"
115   }
116
117   //
118   // Booking data and Payments
119   //
120   group : "booking"
121   {
122     group : "customerdata"  {}
123     {
124       gender : { restrictedTo="male|female" }
125       firstname : { required }
126       lastname : { required }
127
128        group : "address"  {}
129        {
130         street : {}
131         building_no : { semanticTags="ext:buildingno" }
132         zip : { semanticTags="ext:zipcode"}
133         city : {
```

```
134           reactions="tripbooking.booking.customerdata.address.zip:fn_tripbooking....
                  ↪ prefillCity(tripbooking.booking.customerdata.address.zip,$tripbooking....
                  ↪ booking.customerdata.address.city)"
135         }
136         country : {
137           type="country",
138           restrictedTo="fn_tripbooking.getCountries()"
139         }
140         eMail : {
141           semanticTags="ext:email"
142           required
143         }
144         phone : { semanticTags="ext:phone" }
145       }
146     }
147
148     component : "payments"
149     {
150       uri ="http://mimesis.solutions/common/payments/v1"
151       variant ="http://mimesis.solutions/common/payments/v1/variants#root"
152     }
153   }
154
155 }
```

Listing A.3: Anwendungsmodell Kombinierte Reisebuchung

A.8.4 mimesis.model.DSL: Komponente Flugbuchung

Das Listing zeigt eine Flugbuchungs-Komponente zur Erfassung von Anfragedaten für einen Flug mit optionalem Rück- bzw. Gabelflug. Die im Modell enthaltenen Daten besitzen Korrellations-Informationen, sodass aus den erfassten Instanzdaten eine Dienst-Ontologie-Instanz erzeugt werden kann.

```
1  data_description : "flightbooking"
2      uri="http://aviation.mimesis.solutions/products/flightbooking/v1"
3      _version = "1.0.0"
4      referencedOntologies=[
5          base: http://mimesis.soultions/flightbooking/v1#,
6          ext: http://mimesis.solutions/types/ext/v1#,
7          fbo : http://mimesis.soultions/flightbooking/v1#,
8          foaf: http://schema.org/foaf/v1#,
9          xmls: http://www.w3.org/2001/XMLSchema# ]
10 {
11      group : "welcome"
12      { ...    }
13      group : "basictraveldata"
14      {
15          group : "persons"
16          {
17             adults : {
18                 type="number"
19                 min="1"
20                 max="3"
21                 swType="xmls:number"
22                 swForIndividual="flightbookingrequest"
23                 swProperty="fbo:adulttickets" }
24             children : {
25                 type="number"
26                 min="0"
27                 max="3"
28                 swType="xmls:number"
```

```
29                swForIndividual="flightbookingrequest"
30                swProperty="fbo:childtickets"}
31          }
32          group : "preferences"
33          {
34             category : {
35                character="oneOfManyCheck"
36                restrictedTo="firstclass|businessclass|economy"
37                cardinality="1"
38                initialValue="firstclass"
39                swType="xmls:string"
40                swForIndividual="flightbookingrequest"
41                swProperty="fbo:seatcategory"}
42             placement : {
43                character="manyOfManyCheck"
44                restrictedTo="window|passage|isle"
45                initialValue="window|passage|isle"
46                cardinality="*"
47                swType="xmls:string"
48                swForIndividual="flightbookingrequest"
49                swProperty="fbo:seatpreference"}
50             food : {
51                restrictedTo="nopreference|vegetarian|vegan"
52                initialValue="nopreference"
53                swType="xmls:string"
54                swForIndividual="flightbookingrequest"
55                swProperty="fbo:foodpreference" }
56          }
57       }
58       group : "flightinfo"  {
59             cohesion="weak"
60             swClass="fbo:FlightBookingRequest"
61             swIndividual="flightbookingrequest" }
62       {
63          group : "flight"  {
64                swIndividual="flight"
65                swClass="fbo:Flight"
66                swForIndividual="flightbookingrequest"
67                swProperty="fbo:flight" }
68          {
69             startdate : {
70                type="date"
71                swType="xmls:date",
72                swForIndividual="flight"
73                swProperty="fbo:startdate"}
74             fromdestination : {
75                type="text"
76                restrictedTo="fn_flbook.getDepartureAirports()"
77                swForIndividual="flight"
78                swType="xmls:string"
79                swProperty="fbo:fromdestination" }
80             todestination : {
81                restrictedTo=" "
82                cardinality="1"
83                type="text"
84                reactions="flightbooking.flightinfo.flight.fromdestination:fn_flbook....
                      ↪ changeDestinations(flightbooking.flightinfo.flight.fromdestination...
                      ↪ ,$flightbooking.flightinfo.flight.todestination)"
85                activeIf="flightbooking.flightinfo.flight.fromdestination.length>0"
86                swProperty="fbo:todestination"
87                swType="xmls:string"
88                swForIndividual="flight" }
89             returnflight : {
90                type="boolean"
91                initialValue="true" }
92          }
93          group : "returnflight"  {
94                existsIf="(flightbooking.flightinfo.flight.returnflight == true)" }
```

```
        {
           returndate : {
               type="date"
               swType="xmls:date"
               swForIndividual="flight"
               swProperty="fbo:returndate" }
           openjawflight : {
               type="boolean"
               initialValue="false" }
            group : "openjawflightinfo"  {
                  existsIf="(flightbooking.flightinfo.returnflight.openjawflight == true...
                      ↪ )" }
            {
               departuredate : {
                   type="date"
                   swType="xmls:date"
                   swForIndividual="flight"
                   swProperty="fbo:openyawstartdate" }
               openyawfromdestination : {
                   type="text"
                   restrictedTo="fn_flbook.getDepartureAirports()"
                   swType="xmls:string"
                   swForIndividual="flight"
                   swProperty="fbo:openyawfromdestination" }
               openyawtodestination : {
                   type="text"
                   restrictedTo=" "
                   cardinality="1"
                   activeIf="flightbooking.flightinfo.returnflight.openjawflightinfo....
                       ↪ openyawfromdestination.length>0"
                   reactions="flightbooking.flightinfo.returnflight.openjawflightinfo....
                       ↪ openyawfromdestination:fn_flbook.changeDestinations(...
                       ↪ flightbooking.flightinfo.returnflight.openjawflightinfo....
                       ↪ openyawfromdestination,$flightbooking.flightinfo.returnflight....
                       ↪ openjawflightinfo.openyawtodestination)"
                   swType="xmls:string",
                   swForIndividual="flight"
                   swProperty="fbo:openyawtodestination" }
            }
        }
    }

    group : "customerinfo"  {
              swIndividual="customerinfo",
              swClass="fbo:Customerinfo"
              swForIndividual="flightbookingrequest"
              swProperty="fbo:customerinfo" }
    {
        group : "fullname"
        {
           gender : {
               restrictedTo="male|female"
               swType="xmls:string"
               swForIndividual="customerinfo"
               swProperty="foaf:gender" }
           firstname : {
               swType="xmls:string"
               swForIndividual="customerinfo"
               swProperty="foaf:givenname" }
           lastname : {
               required
               swType="xmls:string"
               swForIndividual="customerinfo"
               swProperty="foaf:familyname"     }
        }
        group : "address"  {
              actions="button:click:currentLoaction(flightbooking.customerinfo.address)"
              cohesion="weak"
```

```
157                swIndividual="billingaddress"
158                swClass="fbo:BillingAddress"
159                swForIndividual="customerinfo"
160                swProperty="fbo:billingaddress" }
161            {
162               street : {
163                   swType="xmls:string"
164                   swForIndividual="billingaddress"
165                   swProperty="foaf:street" }
166               building_no : {
167                   semanticTags="ext:buildingno"
168                   swType="xmls:string"
169                   swForIndividual="billingaddress"
170                   swProperty="foaf:buildingNo" }
171               zip : {
172                   semanticTags="ext:zipcode"
173                   swType="xmls:string"
174                   swForIndividual="billingaddress"
175                   swProperty="foaf:zip" }
176               city : {
177                   swType="xmls:string"
178                   swForIndividual="billingaddress"
179                   swProperty="foaf:city" }
180               country : {
181                   type="country"
182                   restrictedTo="germany|austria|swizerland"
183                   swType="xmls:string"
184                   swForIndividual="billingaddress"
185                   swProperty="foaf:country" }
186               email : {
187                   semanticTags="ext:email"
188                   required
189                   swType="xmls:string"
190                   swForIndividual="customerinfo"
191                   swProperty="foaf:email" }
192            }
193            group : "adultpassengers"  {
194                  cohesion="weak"
195                  existsIf="(flightbooking.basictraveldata.persons.adults>0)"}
196            {
197                group : "adult1"  {
198                      existsIf="(flightbooking.basictraveldata.persons.adults>0)" }
199                {
200                   firstname : {
201                       reference="flightbooking.customerinfo.fullname.firstname",
202                       activeIf="(false==true)" }
203                   lastname : {
204                       reference="flightbooking.customerinfo.fullname.lastname",
205                       activeIf="(false==true)" }
206                }
207                group : "adult2"  {  existsIf="(flightbooking.basictraveldata.persons.adults...
                       ↪ >1)"}
208                { ... }
209                group : "adult3"  { existsIf="(flightbooking.basictraveldata.persons.adults...
                       ↪ >2)"}
210                { ... }
211            }
212            group : "youngpassengers"  {
213                  cohesion="weak"
214                  existsIf="(flightbooking.basictraveldata.persons.children>0)"}
215            {
216                group : "child1"  {
217                      existsIf="(flightbooking.basictraveldata.persons.children>0)"}
218                {
219                   firstname : {}
220                   lastname  : {}
221                   age : {
222                       min="1"
```

```
                    max="14"
                    type="number"
                    initialValue="10" }
            }
            group : "child2"  {
                  existsIf="(flightbooking.basictraveldata.persons.children>1)" }
            { ... }
            group : "child3"  {
                  existsIf="(flightbooking.basictraveldata.persons.children>2)" }
            { ... }
        }
    }
...
```

Listing A.4: Anwendungsmodell Flugbuchung

A.8.5 mimesis.variant.DSL: Anpassung der kombinierten Reisebuchung

Die Variantenbeschreibung im folgenden Listing zeigt einen Teil der Anpassungen, die im Rahmen der Erstellung einer Reisebuchung aus Komponenten vorgenommen wurden. Zur Demonstration wurden Komponenten verwendet, deren Komplexität und Umfang stark variieren. Dadurch wurden z.T. umfangreiche Anpassungen erforderlich, andererseits konnten Verknüpfungen und sinnvolle Vorbelegungen von Werten vorgenommen werden, welche die Nutzung der Anwendung vereinfachten (z.B. Vorbelegung von Daten mit dem Reisezeitraum). Diese Änderungen führten zu einem *stattlichen* Variantenmodell, das daher nur auszugsweise dargestellt wird. In der Praxis dürften Kompositionsmodelle einfacher geartet sein.

```
variant_description : {

  //
  // ROOT variant
  //
  variant : "root" {
    uri : "http://bookers.mimesis.solutions/tripbooking/mw2020/variants#root"
    ...
    ////////////////////////////////////
    // customize flightbooking component
    // to use global
    // - first and last name
    // - travellers
    //

    // remove irrelevant sections
    exclude : "tripbooking.options.flightbooking.welcome"
    exclude : "tripbooking.options.flightbooking.save"
    exclude : "tripbooking.options.flightbooking.data"

    // references to components of surrounding modelelements
    modify : {
      type      : "attribute"
      element   : "tripbooking.options.flightbooking.customerinfo.fullname.gender"
      attribute : "reference"
      newvalue  : "tripbooking.basicdata.travellers.gender"
    }
    modify : {
      type      : "attribute"
      element   : "tripbooking.options.flightbooking.customerinfo.fullname.firstname"
      attribute : "reference"
      newvalue  : "tripbooking.basicdata.travellers.firstname"
```

```
33      }
34      modify : {
35        type      : "attribute"
36        element   : "tripbooking.options.flightbooking.customerinfo.fullname.lastname"
37        attribute : "reference"
38        newvalue  : "tripbooking.basicdata.travellers.lastname"
39      }
40      modify : {
41        type      : "attribute"
42        element   : "tripbooking.options.flightbooking.basictraveldata.persons.adults"
43        attribute : "reference"
44        newvalue  : "tripbooking.basicdata.travellers.persons.adults"
45      }
46      modify : {
47        type      : "attribute"
48        element   : "tripbooking.options.flightbooking.basictraveldata.persons.children"
49        attribute : "reference"
50        newvalue  : "tripbooking.basicdata.travellers.persons.children"
51      }
52
53      // ... and hide them in component
54      modify : {
55        type      : "attribute"
56        element   : "tripbooking.options.flightbooking.customerinfo.fullname.gender"
57        attribute : "hidden"
58        newvalue  : "true"
59      }
60      modify : {
61        type      : "attribute"
62        element   : "tripbooking.options.flightbooking.customerinfo.fullname.firstname"
63        attribute : "hidden"
64        newvalue  : "true"
65      }
66      ...
67      modify : {
68        type      : "attribute"
69        element   : "tripbooking.options.flightbooking.basictraveldata.persons.children"
70        attribute : "hidden"
71        newvalue  : "true"
72      }
73
74      //
75      // reference travel dates from surrounding app
76      modify : {
77        type      : "attribute"
78        element   : "tripbooking.options.flightbooking.flightinfo.flight.startdate"
79        attribute : "reference"
80        newvalue  : "tripbooking.basicdata.traveldates.startdate"
81      }
82      modify : {
83        type      : "attribute"
84        element   : "tripbooking.options.flightbooking.flightinfo.returnflight.returndate"
85        attribute : "reference"
86        newvalue  : "tripbooking.basicdata.traveldates.enddate"
87      }
88      // ... and deactivate manipulation in component
89      modify : {
90        type      : "attribute"
91        element   : "tripbooking.options.flightbooking.flightinfo.flight.startdate"
92        attribute : "activeIf"
93        newvalue  : "false"
94      }
95      modify : {
96        type      : "attribute"
97        element   : "tripbooking.options.flightbooking.flightinfo.returnflight.returndate"
98        attribute : "activeIf"
99        newvalue  : "false"
100     }
```

```
101     // reference travel destinations from surrounding app
102     ...
103     // customize payment data
104     // reference values from environment
105     ...
106     // customize customer data (travelgroup)
107     // reference values from environment
108     ...
109
110     ////////////////////////////////
111     // customize carrental component
112     //
113     // seamlessly integrate the options in flow of questions ,
114     // not as separate displyunits as originally suggested
115     modify : {
116       type      : "attribute"
117       modification: "remove"
118       element   : "tripbooking.options.carrental.car"
119       attribute : "cohesion"
120     }
121
122     // set initialValues from surrounding application
123     // NOT REFERENCES, just the initial value
124     modify : {
125       type      : "attribute"
126       element   : "tripbooking.options.carrental.car.pickuptime"
127       attribute : "initialValue"
128       newvalue  : "mimesis.base.getFrom(tripbooking.basicdata.traveldates.startdate)"
129     }
130     modify : {
131       type      : "attribute"
132       element   : "tripbooking.options.carrental.car.returntime"
133       attribute : "initialValue"
134       newvalue  : "mimesis.base.getFrom(tripbooking.basicdata.traveldates.enddate)"
135     }
136
137     //
138     // modify payment data
139     // reference values from environment
140     ...
141     //
142     // modify customer data
143     // reference values from environment
144     ...
145
146     ///////////////////////
147     // modify babysitting component
148     //
149
150     // seamlessly integrate the options, not as separate displyunits
151     modify : {
152       type      : "attribute"
153       modification: "remove"
154       element   : "tripbooking.options.babysitter.data"
155       attribute : "cohesion"
156     }
157     // preset dates times ...
158     modify : {
159       type      : "attribute"
160       element   : "tripbooking.options.babysitter.data.babysitterdetails.pickuptime"
161       attribute : "initialValue"
162       newvalue  : "mimesis.base.getFrom(tripbooking.basicdata.traveldates.startdate)"
163     }
164
165     ///////////////////////
166     // modify overnight stays component
167     //
168
```

```
169     // seamlessly integrate the options, not as separate displyunits
170     ...
171
172     ////////////////////////
173     // modify events component
174     //
175
176     // seamlessly integrate the options, not as separate displyunits
177     // preset dates/times
178     ...
179
180     ////////////////////////////
181     // customize insurance component
182     //
183
184     // seamlessly integrate the options, not as separate displyunits
185     modify : {
186       type      : "attribute"
187       modification: "remove"
188       element   : "tripbooking.insurance"
189       attribute : "cohesion"
190     }
191     modify : {
192       type      : "attribute"
193       modification: "remove"
194       element   : "tripbooking.insurance.coverage"
195       attribute : "cohesion"
196     }
197     modify : {
198       type      : "attribute"
199       modification: "set"
200       element   : "tripbooking.insurance.customerdata"
201       attribute : "hidden"
202       newvalue  : "true"
203     }
204
205     ////////////////////////
206     // modify general booking and payments
207     //
208
209     // seamlessly integrate the options, not as separate displyunits
210     modify : {
211       type      : "attribute"
212       modification: "remove"
213       element   : "tripbooking.booking.payments.paymentdata"
214       attribute : "cohesion"
215     }
216
217     // reference gender, firstname, lastname and deactivate
218     modify : {
219       type      : "attribute"
220       element   : "tripbooking.booking.customerdata.gender"
221       attribute : "reference"
222       newvalue  : "tripbooking.basicdata.travellers.gender"
223     }
224     ...
225
226     // activation
227     modify : {
228       type      : "attribute"
229       element   : "tripbooking.booking.customerdata.gender"
230       attribute : "activeIf"
231       newvalue  : "false"
232     }
233    ...
234
235  }
236
```

```
237   //
238   // CUSTOMER-variant
239   //
240   variant : "customer" {
241     uri : "http://bookers.mimesis.solutions/tripbooking/mw2020/variants#customer"
242     extends:"http://bookers.mimesis.solutions/tripbooking/mw2020/variants#root"
243
244     // no open yaw flights
245     exclude : "tripbooking.options.flightbooking.flightinfo.returnflight.openjawflight"
246     exclude : "tripbooking.options.flightbooking.flightinfo.returnflight....
          ↪ openjawflightinfo"
247
248     // REMOVE OPTIONS from selection in this variant
249     exclude : "tripbooking.options.babysitter"
250     exclude : "tripbooking.options.concert"
251     modify : {
252       type      : "attribute"
253       element   : "tripbooking.options.components"
254       attribute : "restrictedTo"
255       newvalue  : "flight|car|overnightstay"
256     }
257
258     // CHANGE INTERACTIONELEMENT CHARACTER
259     modify : {
260       type      : "attribute"
261       element   : "tripbooking.basicdata.destinations.destination"
262       attribute : "character"
263       newvalue  : "iconicSelect"
264     }
265     // Flightbooking customisation
266     // set returnflight as fixed option
267     modify : {
268       type      : "attribute"
269       element   : "tripbooking.options.flightbooking.flightinfo.flight.returnflight"
270       attribute : "initialValue"
271       newvalue  : "true"
272     }
273     modify : {
274       type      : "attribute"
275       element   : "tripbooking.options.flightbooking.flightinfo.flight.returnflight"
276       attribute : "hidden"
277       newvalue  : "true"
278     }
279     ...
280   }
281 }
```

Listing A.5: Variantenbeschreibung der kombinierten Reisebuchung

A.8.6 mimesis.model.OWL: Ontologie Komponente Flugbuchung

Die folgenden Listings zeigen den Auszug der OWL-Ontologie für eine Flugbuchungs-Komponente in JSONLD-Notation. Listing A.6 zeigt die Header-Informationen, verwendeten Profil-Annotationen, sowie die Klassen und Relations-Deklarationen. Listing A.7 zeigt die Deklaration der Datenelemente, die Klassen zugeordnet sind. Die beschriebenen Daten besitzen Korrelations-Informationen, um aus den erfassten Instanzdaten eine Dienst-Ontologie-Instanz herleiten zu können.

```
@prefix : <http://aviation.mimesis.solutions/products/flightbooking/v1#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> @base <http://aviation.mimesis....
    ↪ solutions/products/flightbooking/v1> .
<http://aviation.mimesis.solutions/products/flightbooking/v1> rdf:type owl:Ontology .

#################################################################
#    Used Profile annotations
#################################################################
<http://mimesis.solutions/annotations#actions> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#activeIf> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#cardinality> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#character> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#cohesion> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#existsIf> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#initialValue> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#reactions> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#reference> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#required> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#restrictedTo> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#semanticTags> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#sequence> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#swClass> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#swForIndividual> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#swIndividual> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#swProperty> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#swType> rdf:type owl:AnnotationProperty
<http://mimesis.solutions/annotations#type> rdf:type owl:AnnotationProperty

#################################################################
#    Classes
#################################################################
:Address rdf:type owl:Class .
:Adult1 rdf:type owl:Class .
:Adult2 rdf:type owl:Class .
:Adult3 rdf:type owl:Class .
:Adultpassengers rdf:type owl:Class .
:Basictraveldata rdf:type owl:Class .
:Child1 rdf:type owl:Class .
:Child2 rdf:type owl:Class .
:Child3 rdf:type owl:Class .
:Customer rdf:type owl:Class .
:Customerinfo rdf:type owl:Class .
:Data rdf:type owl:Class .
:Flight rdf:type owl:Class .
:Flightbooking rdf:type owl:Class .
:Flightinfo rdf:type owl:Class .
:Fullname rdf:type owl:Class .
:Openjawflightinfo rdf:type owl:Class .
:Passengerinfo rdf:type owl:Class .
:Persons rdf:type owl:Class .
:Preferences rdf:type owl:Class .
:Returnflight rdf:type owl:Class .
```

```
:Youngpassengers rdf:type owl:Class

#################################################################
#    Object Properties (Relations between classes)
#################################################################
:Adultpassengers.adult1 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Adultpassengers ;
      rdfs:range :Adult1 ;
      <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
          ↪ persons.adults>0)" ;
      <http://mimesis.solutions/annotations#sequence> "1" .
:Adultpassengers.adult2 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Adultpassengers ;
      rdfs:range :Adult2 ;
      <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
          ↪ persons.adults>1)" ;
      <http://mimesis.solutions/annotations#sequence> "2" .
:Adultpassengers.adult3 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Adultpassengers ;
      rdfs:range :Adult3 ;
      <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
          ↪ persons.adults>2)" ;
      <http://mimesis.solutions/annotations#sequence> "3" .
:Basictraveldata.persons rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Basictraveldata ;
      rdfs:range :Persons ;
      <http://mimesis.solutions/annotations#sequence> "1" .
:Basictraveldata.preferences rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Basictraveldata ;
      rdfs:range :Preferences ;
      <http://mimesis.solutions/annotations#sequence> "2" .
:Customerinfo.address rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Customerinfo ;
      rdfs:range :Address ;
      <http://mimesis.solutions/annotations#actions> "button:click:currentLoaction(...
          ↪ flightbooking.customerinfo.address)" ;
      <http://mimesis.solutions/annotations#cohesion> "weak" ;
      <http://mimesis.solutions/annotations#sequence> "2" ;
      <http://mimesis.solutions/annotations#swClass> "fbo:BillingAddress" ;
      <http://mimesis.solutions/annotations#swForIndividual> "customerinfo" ;
      <http://mimesis.solutions/annotations#swIndividual> "billingaddress" ;
      <http://mimesis.solutions/annotations#swProperty> "fbo:billingaddress" .
:Customerinfo.fullname rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Customerinfo ;
      rdfs:range :Fullname ;
      <http://mimesis.solutions/annotations#sequence> "1" .
:Flightbooking.basictraveldata rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Flightbooking ;
      rdfs:range :Basictraveldata ;
      <http://mimesis.solutions/annotations#sequence> "1" .
:Flightbooking.customerinfo rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Flightbooking ;
      rdfs:range :Customerinfo ;
      <http://mimesis.solutions/annotations#sequence> "4" ;
      <http://mimesis.solutions/annotations#swClass> "fbo:Customerinfo" ;
      <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
      <http://mimesis.solutions/annotations#swIndividual> "customerinfo" ;
      <http://mimesis.solutions/annotations#swProperty> "fbo:customerinfo" .
:Flightbooking.data rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Flightbooking ;
      rdfs:range :Data ;
      <http://mimesis.solutions/annotations#sequence> "6" .
:Flightbooking.flightinfo rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Flightbooking ;
      rdfs:range :Flightinfo ;
      <http://mimesis.solutions/annotations#cohesion> "weak" ;
      <http://mimesis.solutions/annotations#sequence> "2" ;
      <http://mimesis.solutions/annotations#swClass> "fbo:FlightBookingRequest" ;
```

```
120       <http://mimesis.solutions/annotations#swIndividual> "flightbookingrequest" .
121 :Flightbooking.passengerinfo rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
122       rdfs:domain :Flightbooking ;
123       rdfs:range :Passengerinfo ;
124       <http://mimesis.solutions/annotations#sequence> "3" ;
125       <http://mimesis.solutions/annotations#swClass> "fbo:Passengerinfo" ;
126       <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
127       <http://mimesis.solutions/annotations#swIndividual> "passengerinfo" ;
128       <http://mimesis.solutions/annotations#swProperty> "fbo:passengerinfo" .
129 :Flightinfo.flight rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
130       rdfs:domain :Flightinfo ;
131       rdfs:range :Flight ;
132       <http://mimesis.solutions/annotations#sequence> "1" ;
133       <http://mimesis.solutions/annotations#swClass> "fbo:Flight" ;
134       <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
135       <http://mimesis.solutions/annotations#swIndividual> "flight" ;
136       <http://mimesis.solutions/annotations#swProperty> "fbo:flight" .
137 :Flightinfo.returnflight rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
138       rdfs:domain :Flightinfo ;
139       rdfs:range :Returnflight ;
140       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.flightinfo.flight....
              ↪ returnflight == true)" ;
141       <http://mimesis.solutions/annotations#sequence> "2" .
142 :Passengerinfo.adultpassengers rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
143       rdfs:domain :Passengerinfo ;
144       rdfs:range :Adultpassengers ;
145       <http://mimesis.solutions/annotations#cohesion> "weak" ;
146       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
              ↪ persons.adults>0)" ;
147       <http://mimesis.solutions/annotations#sequence> "1" .
148 :Passengerinfo.youngpassengers rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
149       rdfs:domain :Passengerinfo ;
150       rdfs:range :Youngpassengers ;
151       <http://mimesis.solutions/annotations#cohesion> "weak" ;
152       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
              ↪ persons.children>0)" ;
153       <http://mimesis.solutions/annotations#sequence> "2" .
154 :Returnflight.openjawflightinfo rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
155       rdfs:domain :Returnflight ;
156       rdfs:range :Openjawflightinfo ;
157       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.flightinfo....
              ↪ returnflight.openjawflight == true)" ;
158       <http://mimesis.solutions/annotations#sequence> "3" .
159 :Youngpassengers.child1 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
160       rdfs:domain :Youngpassengers ;
161       rdfs:range :Child1 ;
162       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
              ↪ persons.children>0)" ;
163       <http://mimesis.solutions/annotations#sequence> "1" .
164 :Youngpassengers.child2 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
165       rdfs:domain :Youngpassengers ;
166       rdfs:range :Child2 ;
167       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
              ↪ persons.children>1)" ;
168       <http://mimesis.solutions/annotations#sequence> "2" .
169 :Youngpassengers.child3 rdf:type owl:ObjectProperty , owl:FunctionalProperty ;
170       rdfs:domain :Youngpassengers ;
171       rdfs:range :Child3 ;
172       <http://mimesis.solutions/annotations#existsIf> "(flightbooking.basictraveldata....
              ↪ persons.children>2)" ;
173       <http://mimesis.solutions/annotations#sequence> "3" .
```

Listing A.6: Ontologie Flugbuchung: Klassen und Beziehungen

```
1
2     ################################################################
3     #    Data properties
4     ################################################################
5
6     :Address.street rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
7           rdfs:domain :Address ;
8           rdfs:range <http://mimesis.solutions/datatypes#text> ;
9           <http://mimesis.solutions/annotations#sequence> "1" ;
10          <http://mimesis.solutions/annotations#swForIndividual> "billingaddress" ;
11          <http://mimesis.solutions/annotations#swProperty> "foaf:street" ;
12          <http://mimesis.solutions/annotations#swType> "xmls:string" .
13    :Address.building_no rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
14          rdfs:domain :Address ;
15          rdfs:range <http://mimesis.solutions/datatypes#text> ;
16          <http://mimesis.solutions/annotations#semanticTags> "ext:buildingno" ;
17          <http://mimesis.solutions/annotations#sequence> "2" ;
18          <http://mimesis.solutions/annotations#swForIndividual> "billingaddress" ;
19          <http://mimesis.solutions/annotations#swProperty> "foaf:buildingNo" ;
20          <http://mimesis.solutions/annotations#swType> "xmls:string" .
21    :Address.city rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
22          rdfs:domain :Address ;
23          rdfs:range <http://mimesis.solutions/datatypes#text> ;
24          <http://mimesis.solutions/annotations#sequence> "4" ;
25          <http://mimesis.solutions/annotations#swForIndividual> "billingaddress" ;
26          <http://mimesis.solutions/annotations#swProperty> "foaf:city" ;
27          <http://mimesis.solutions/annotations#swType> "xmls:string" .
28    :Address.zip rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
29          ...
30    :Address.country rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
31          ...
32    :Address.email rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
33          ...
34    :Adult1.firstname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
35          rdfs:domain :Adult1 ;
36          rdfs:range <http://mimesis.solutions/datatypes#text> ;
37          <http://mimesis.solutions/annotations#reference> "flightbooking.customerinfo....
                ↪ fullname.firstname" ;
38          <http://mimesis.solutions/annotations#sequence> "1" .
39    :Adult1.lastname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
40          rdfs:domain :Adult1 ;
41          rdfs:range <http://mimesis.solutions/datatypes#text> ;
42          <http://mimesis.solutions/annotations#reference> "flightbooking.customerinfo....
                ↪ fullname.lastname" ;
43          <http://mimesis.solutions/annotations#sequence> "2"
44    ### ... Adult 2 and 3 ommitted
45    :Child1.age rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
46          rdfs:domain :Child1 ;
47          rdfs:range xsd:integer ;
48          <http://mimesis.solutions/annotations#initialValue> "10" ;
49          <http://mimesis.solutions/annotations#max> "14" ;
50          <http://mimesis.solutions/annotations#min> "1" ;
51          <http://mimesis.solutions/annotations#sequence> "3" ;
52          <http://mimesis.solutions/annotations#type> "number" .
53    :Child1.firstname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
54         ...
55          <http://mimesis.solutions/annotations#sequence> "1" .
56    :Child1.lastname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
57         ...
58          <http://mimesis.solutions/annotations#sequence> "2"
59    ### ... Child 2 and 3 ommitted
60
61    :Flight.fromdestination rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
62          rdfs:domain :Flight ;
63          rdfs:range xsd:string ;
64          <http://mimesis.solutions/annotations#restrictedTo> "fn_flbook....
                ↪ getDepartureAirports()" ;
65          <http://mimesis.solutions/annotations#sequence> "2" ;
```

```
66          <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
67          <http://mimesis.solutions/annotations#swProperty> "fbo:fromdestination" ;
68          <http://mimesis.solutions/annotations#swType> "xmls:string" ;
69          <http://mimesis.solutions/annotations#type> "text" .
70    :Flight.returnflight rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
71          rdfs:domain :Flight ;
72          rdfs:range <http://mimesis.solutions/datatypes#boolean> ;
73          <http://mimesis.solutions/annotations#initialValue> "true" ;
74          <http://mimesis.solutions/annotations#sequence> "4" ;
75          <http://mimesis.solutions/annotations#type> "boolean" .
76    :Flight.startdate rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
77          rdfs:domain :Flight ;
78          rdfs:range xsd:date ;
79          <http://mimesis.solutions/annotations#sequence> "1" ;
80          <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
81          <http://mimesis.solutions/annotations#swProperty> "fbo:startdate" ;
82          <http://mimesis.solutions/annotations#swType> "xmls:date" ;
83          <http://mimesis.solutions/annotations#type> "date" .
84    :Flight.todestination rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
85          rdfs:domain :Flight ;
86          rdfs:range xsd:string ;
87          <http://mimesis.solutions/annotations#activeIf> "flightbooking.flightinfo....
                ↪ flight.fromdestination.length>0" ;
88          <http://mimesis.solutions/annotations#cardinality> "1" ;
89          <http://mimesis.solutions/annotations#reactions> "flightbooking.flightinfo....
                ↪ flight.fromdestination:fn_flbook.changeDestinations(...)" ;
90          <http://mimesis.solutions/annotations#restrictedTo> " " ;
91          <http://mimesis.solutions/annotations#sequence> "3" ;
92          <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
93          <http://mimesis.solutions/annotations#swProperty> "fbo:todestination" ;
94          <http://mimesis.solutions/annotations#swType> "xmls:string" ;
95          <http://mimesis.solutions/annotations#type> "text" .
96    :Fullname.firstname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
97          rdfs:domain :Fullname ;
98          rdfs:range <http://mimesis.solutions/datatypes#text> ;
99          <http://mimesis.solutions/annotations#sequence> "2" ;
100         <http://mimesis.solutions/annotations#swForIndividual> "customerinfo" ;
101         <http://mimesis.solutions/annotations#swProperty> "foaf:givenname" ;
102         <http://mimesis.solutions/annotations#swType> "xmls:string" .
103   :Fullname.gender rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
104         rdfs:domain :Fullname ;
105         rdfs:range <http://mimesis.solutions/datatypes#text> ;
106         <http://mimesis.solutions/annotations#restrictedTo> "male|female" ;
107         <http://mimesis.solutions/annotations#sequence> "1" ;
108         <http://mimesis.solutions/annotations#swForIndividual> "customerinfo" ;
109         <http://mimesis.solutions/annotations#swProperty> "foaf:gender" ;
110         <http://mimesis.solutions/annotations#swType> "xmls:string" .
111   :Fullname.lastname rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
112         rdfs:domain :Fullname ;
113         rdfs:range <http://mimesis.solutions/datatypes#text> ;
114         <http://mimesis.solutions/annotations#required> "" ;
115         <http://mimesis.solutions/annotations#sequence> "3" ;
116         <http://mimesis.solutions/annotations#swForIndividual> "customerinfo" ;
117         <http://mimesis.solutions/annotations#swProperty> "foaf:familyname" ;
118         <http://mimesis.solutions/annotations#swType> "xmls:string" .
119   :Openjawflightinfo.departuredate rdf:type owl:DatatypeProperty , owl:...
        ↪ FunctionalProperty ;
120         rdfs:domain :Openjawflightinfo ;
121         rdfs:range xsd:date ;
122         <http://mimesis.solutions/annotations#sequence> "1" ;
123         <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
124         <http://mimesis.solutions/annotations#swProperty> "fbo:openyawstartdate" ;
125         <http://mimesis.solutions/annotations#swType> "xmls:date" ;
126         <http://mimesis.solutions/annotations#type> "date" .
127   :Openjawflightinfo.openyawfromdestination rdf:type owl:DatatypeProperty , owl:...
        ↪ FunctionalProperty ;
128         rdfs:domain :Openjawflightinfo ;
129         rdfs:range xsd:string ;
```

```
130         <http://mimesis.solutions/annotations#restrictedTo> "fn_flbook....
                ↪ getDepartureAirports()" ;
131         <http://mimesis.solutions/annotations#sequence> "2" ;
132         <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
133         <http://mimesis.solutions/annotations#swProperty> "fbo:openyawfromdestination" ...
                ↪ ;
134         <http://mimesis.solutions/annotations#swType> "xmls:string" ;
135         <http://mimesis.solutions/annotations#type> "text" .
136     :Openjawflightinfo.openyawtodestination rdf:type owl:DatatypeProperty , owl:...
          ↪ FunctionalProperty ;
137         rdfs:domain :Openjawflightinfo ;
138         rdfs:range xsd:string ;
139         <http://mimesis.solutions/annotations#activeIf> "flightbooking.flightinfo....
                ↪ returnflight.openjawflightinfo.openyawfromdestination.length>0" ;
140         <http://mimesis.solutions/annotations#cardinality> "1" ;
141         <http://mimesis.solutions/annotations#reactions> "flightbooking.flightinfo....
                ↪ returnflight.openjawflightinfo.openyawfromdestination:fn_flbook....
                ↪ changeDestinations(...)" ;
142         <http://mimesis.solutions/annotations#restrictedTo> " " ;
143         <http://mimesis.solutions/annotations#sequence> "3" ;
144         <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
145         <http://mimesis.solutions/annotations#swProperty> "fbo:openyawtodestination" ;
146         <http://mimesis.solutions/annotations#swType> "xmls:string" ;
147         <http://mimesis.solutions/annotations#type> "text" .
148     :Persons.adults rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
149         rdfs:domain :Persons ;
150         rdfs:range xsd:integer ;
151         <http://mimesis.solutions/annotations#max> "3" ;
152         <http://mimesis.solutions/annotations#min> "1" ;
153         <http://mimesis.solutions/annotations#sequence> "1" ;
154         <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
155         <http://mimesis.solutions/annotations#swProperty> "fbo:adulttickets" ;
156         <http://mimesis.solutions/annotations#swType> "xmls:number" ;
157         <http://mimesis.solutions/annotations#type> "number" .
158     :Persons.children rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
159         rdfs:domain :Persons ;
160         rdfs:range xsd:integer ;
161         <http://mimesis.solutions/annotations#max> "3" ;
162         <http://mimesis.solutions/annotations#min> "0" ;
163         <http://mimesis.solutions/annotations#sequence> "2" ;
164         <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
165         <http://mimesis.solutions/annotations#swProperty> "fbo:childtickets" ;
166         <http://mimesis.solutions/annotations#swType> "xmls:number" ;
167         <http://mimesis.solutions/annotations#type> "number" .
168     :Preferences.category rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
169         rdfs:domain :Preferences ;
170         rdfs:range <http://mimesis.solutions/datatypes#text> ;
171         <http://mimesis.solutions/annotations#cardinality> "1" ;
172         <http://mimesis.solutions/annotations#character> "oneOfManyCheck" ;
173         <http://mimesis.solutions/annotations#initialValue> "economy" ;
174         <http://mimesis.solutions/annotations#restrictedTo> "firstclass|businessclass|...
                ↪ economy" ;
175         <http://mimesis.solutions/annotations#sequence> "1" ;
176         <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
177         <http://mimesis.solutions/annotations#swProperty> "fbo:seatcategory" ;
178         <http://mimesis.solutions/annotations#swType> "xmls:string" .
179     :Preferences.food rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
180         rdfs:domain :Preferences ;
181         rdfs:range <http://mimesis.solutions/datatypes#text> ;
182         <http://mimesis.solutions/annotations#initialValue> "nopreference" ;
183         <http://mimesis.solutions/annotations#restrictedTo> "nopreference|vegetarian|...
                ↪ vegan" ;
184         <http://mimesis.solutions/annotations#sequence> "3" ;
185         <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
186         <http://mimesis.solutions/annotations#swProperty> "fbo:foodpreference" ;
187         <http://mimesis.solutions/annotations#swType> "xmls:string" .
188     :Preferences.placement rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
189         rdfs:domain :Preferences ;
```

```
190           rdfs:range <http://mimesis.solutions/datatypes#text> ;
191           <http://mimesis.solutions/annotations#cardinality> "*" ;
192           <http://mimesis.solutions/annotations#character> "manyOfManyCheck" ;
193           <http://mimesis.solutions/annotations#initialValue> "window|passage|isle" ;
194           <http://mimesis.solutions/annotations#restrictedTo> "window|passage|isle" ;
195           <http://mimesis.solutions/annotations#sequence> "2" ;
196           <http://mimesis.solutions/annotations#swForIndividual> "flightbookingrequest" ;
197           <http://mimesis.solutions/annotations#swProperty> "fbo:seatpreference" ;
198           <http://mimesis.solutions/annotations#swType> "xmls:string" .
199     :Returnflight.openjawflight rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
200           rdfs:domain :Returnflight ;
201           rdfs:range <http://mimesis.solutions/datatypes#boolean> ;
202           <http://mimesis.solutions/annotations#initialValue> "false" ;
203           <http://mimesis.solutions/annotations#sequence> "2" ;
204           <http://mimesis.solutions/annotations#type> "boolean" .
205     :Returnflight.returndate rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
206           rdfs:domain :Returnflight ;
207           rdfs:range xsd:date ;
208           <http://mimesis.solutions/annotations#sequence> "1" ;
209           <http://mimesis.solutions/annotations#swForIndividual> "flight" ;
210           <http://mimesis.solutions/annotations#swProperty> "fbo:returndate" ;
211           <http://mimesis.solutions/annotations#swType> "xmls:date" ;
212           <http://mimesis.solutions/annotations#type> "date" .
```

Listing A.7: Ontologie Flugbuchung: Datenelemente und Attribute

B. Verzeichnisse

B.1 Abbildungsverzeichnis

B.2 Tabellenverzeichnis

B.3 Listings

B.4 Abkürzungsverzeichnis

AO	Applikations-Ontologie (Application Ontology)
AOI	Applikations-Ontologie-Instanz (Application Ontology Instance)
API	Advanced Programming Interface
AUI	Abstract User Interface
CRUD	Create Read Update Delete
CUI	Concrete UI
DO	Dienstontologie (Domain Ontology)
DOI	Dienstontologie-Instanz (Domain Ontology Instance)
DOM	Document Object Model
MDSD	Model-Driven Software Development
DSL	Domain Specific Language
DSR	Design Science Research
FUI	Final User Inetrface
GUI	Graphical User Interface
HTML	HypertextMarkup Language
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier
IT	Informationstechnologie
IoE	Internet of Everything
IoT	Internet of Things
JSF	Java Server Faces
JSON	JavaScript Object Notation
JSONLD	JSON for Linking Data
JavaAWT	Abstract Window Toolkit
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Enineering
MVC	Model View Controller
MVVM	Model View Viewmodel
OO	Object Orientation
OWL	Web Ontology Language
PIM	Platform independent Model
PSM	Platform Specific Model
QoS	Quality Of Service
RDF	Resource Description Framework
REST	Representational State Transfer
SOA	Service Oriented Architecture
SOAP	Simple Objec Access Protocol
TOI	Target Ontology Instance
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UIDL	User Interface Description Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
WIMP	Windows, Icons, Menus und Pointer
WSDL	Web Service Description Language
XIB	XML Interface Builder
XML	Extensible Markup Language
XSD	XML Schema Definition

D. Persönliche Veröffentlichungen

- Hitz, M. et al., 2018:
  Automatic UI Generation for Aggregated Linked Data Applications by Using Sharable Application Ontologies.
  In L. F. Pires, S. Hammoudi, & B. Selic, eds. Model-Driven Engineering and Software Development - 5th International Conference, MODELSWARD 2017, Porto, Portugal, February 19-21, 2017, Revised Selected Papers. Communications in Computer and Information Science. Vol. 880. Springer, Cham pp. 328–353.

- Hitz, M., Kessel, T. & Pfisterer, D., 2017:
  Towards Sharable Application Ontologies for the Automatic Generation of UIs for Dialog based Linked Data Applications.
  In Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017), SCITEPRESS – Science and Technology Publications, Lda., pp. 65-77.

- Hitz, M. & Kessel, T., 2016:
  Using Application Ontologies for the Automatic Generation of User Interfaces for Dialog-Based Applications Data-centric Description of Interview Applications.
  In Proceedings of CONFENIS 2016, International Conference on Research and Practical Issues of Enterprise Information Systems, LNBIP 268. Springer International Publishing AG, pp. 16–31.

- Hitz, M. et al., 2016:
  Generic UIs for Requesting Complex Products within Distributed Market Spaces in the Internet of Everything.
  In F. Buccafurri et al., eds. Proceedings of CD-ARES 2016, Availability, Reliability and Security in Information Systems, LNCS 9817. Springer International Publishing, pp. 29–44.

- Hitz, M., 2016:
  mimesis : Ein datenzentrierter Ansatz zur Modellierung von Varianten für Interview-Anwendungen.
  In V. Nissen et al., eds., Proceedings - Multikonferenz Wirtschaftsinformatik (MKWI) 2016. Ilmenau: TU Ilmenau Universitätsbibliothek, pp. 1155–1165.

- Hitz, M., 2013:
  Konzept zur Variantenbildung von Oberflächen in einer Multikanal-Umgebung.
  In Lecture Notes in Informatics INFORMATIK 2013 - GI-Jahrestagung 2013, LNI 220. GI, Köllen Druck+Verlag GmbH, Bonn., pp. 2664–2678.

- Hitz, M., 2013:
  Eine Multikanal-Architektur für Frontendsysteme und deren Erweiterbarkeit durch Variantenbildung.
  In Wagner, S. & Lichter, H. (Hrsg.), Proceedings Software Engineering 2013, Workshopband, LNI 215. GI, Köllen Druck+Verlag GmbH, Bonn., pp. 583–589.

- Hitz, M. & Kessel, T., 2013:
  Einfluss der Nutzung und Auswahl von Open Source Software beim Entwurf einer Multikanal-Architektur.
  In Lecture Notes in Informatics INFORMATIK 2013 - GI-Jahrestagung 2013, LNI 220. pp. 1324–1338.

Literatur

[1] Silvia Abrahao, Lionel Balme, Niels Ole Bernsen, Laurent Bouillon, Gaëlle Calvary, Joëlle Coutaz, Quentin Limbourg, und Et. *UsiXML Reference Manual (V1.8), USer Interface eXtensible Markup Language*. February. Université catholique de Louvain (UCL) Louvain. 2007.

[2] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, und Jonathan E. Shuster. "UIML: An Appliance-Independent XML User Interface Language". In *WWW '99 Proceedings of the Eighth International Conference on World Wide Web*, 1695–1708. 1999.

[3] Mohammed AbuJarour, Mircea Craculeac, Falko Menge, Tobias Vogel, und Jan Felix Schwarz. "Posr: A Comprehensive System for Aggregating and Using Web Services". *SERVICES 2009 - 5th 2009 World Congress on Services*, Nummer PART 1: 139–146. 2009. doi:10.1109/SERVICES-I.2009.111.

[4] Marzia Adorni, Franceses Arcelli, Claudia Raibulet, Marcello Sarini, und Francesco Tisato. "Designing an Architecture for Multichannel Adaptive Information Systems". *Proceedings of the International Conference on Software Engineering Research and Practice, SERP'04* 2: 652–658. 2004.

[5] Jean Vanderdonckt, Benjamin Michotte}, Francisco Montero Adrian Stanciulescu Quentin Limbourg. "A Transformational Approach for Multimodal Web User Interfaces Based on UsiXML". In *ICMI '05: Proceedings of the 7th International Conference on Multimodal Interfaces*, 259–266. 2005.

[6] J.E. van Aken. "Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules". *Journal of Management Studies* 41 (2): 219–246. 2004.

[7] Mir Farooq Ali, Manuel A Pérez-quiñones, Eric Shell, Virginia Tech, und Marc Abrams. "Building Multi-Platform User Interfaces with UIML". In *Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces*, 255–265. March. 2002.

[8] Vladimir L. Averbukh, Mihkail O. Bakhterev, Aleksandr Yu. Baydalin, Dmitriy Yu. Gorbashevskiy, Damir R. Ismagilov, Alexey Yu. Kazantsev, Polina V. Nebogatikova,

Anna V. Popova, und Pavel A. Vasev. “Searching and Analysis of Interface and Visualization Metaphors”. In *Human Computer Interaction*, herausgegeben von Kikuo Asai, Kapitel Ch. 3. InTech, Rijeka. 2008. doi:10.5772/5880.

[9] Helmut Balzert. “From OOA to GUI - The JANUS System”. In *Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction*, herausgegeben von Helmersen, P.H., Gilmore, D.J., Arnesen}, S.A. (eds.) Nordbyn K., 319–325. 1995.

[10] Helmut Balzert, Frank Hofmann, und Volker Kruschinski. “The JANUS Application Development Environment—Generating More than the User Interface”. In *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces (CADUI’1996)*, 183–206. 1996.

[11] Frank Bannister. “Dismantling the Silos: Extracting New Value from IT Investments in Public Administration”. *Information Systems Journal* 11: 65–84. 2001.

[12] Lina Barakat, Simon Miles, und Michael Luck. “Efficient Correlation-Aware Service Selection”. In *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 1–8. IEEE. 2012. doi:10.1109/ICWS.2012.62.

[13] J.A. Bargas-Avila, A.N. Tuch, K. Opwis, S.P Roth, O. Brenzikofer, und S. Orsini. “Simple but Crucial User Interfaces in the World Wide Web: Introducing 20 Guidelines for Usable Web Form Design”. In *User Interfaces*, herausgegeben von Rita Matrai, InTech Ope, 1–11, Kapitel 1. May. InTech. 2010. doi:10.5772/9500.

[14] Alistair P. Barros, und Marlon Dumas. “The Rise of Web Service Ecosystems”. *IT Professional* 8 (5): 31–37. 2006. doi:10.1109/MITP.2006.123.

[15] Harold Boley, und Elizabeth Chang. “Digital Ecosystems: Principles and Semantics”. In *2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007) Digital*, 398–403. 2007. doi:10.1109/DEST.2007.372005.

[16] Thomas Bosch, Erman Acar, Andreas Nolle, und Kai Eckert. “The Role of Reasoning for RDF Validation”. In *Semantics 2015 : Proceedings of the 11th International Conference on Semantic Systems, Vienna, Austria, September 15-17, 2015*, 33–40. ACM. 2015. doi:10.1145/2814864.2814867.

[17] Rachel Botsman, und Roo Rogers. *What’s Mine Is Yours: The Rise of Collaborative Consumption*. Harper Business, New York. 2010.

[18] Goetz Botterweck. “A Model-Driven Approach to the Engineering of Multiple User Interfaces”. In *Proceeding MoDELS’06 Proceedings of the 2006 International Conference on Models in Software Engineering*, 106–115. 2006.

[19] Jens Böcker. “Die Customer Journey - Chance Für Mehr Kundennähe”. In *Dialogmarketing Perspektiven 2014/2015: Tagungsband 9. Wissenschaftlicher Interdisziplinärer Kongress Für Dialogmarketing*, 165–177. Springer Fachmedien Wiesbaden, Wiesbaden. 2015. doi:10.1007/978-3-658-08876-7_8.

[20] Joseph L. Bower, und Clayton M. Christensen. “Disruptive Technologies: Catching the Wave.” *Harvard Business Review* 73 (1): 43–53. 1995. https://hbr.org/1995/01/disruptive-technologies-catching-the-wave.

[21] G. Briscoe, und P. De Wilde. *Digital Ecosystems: Evolving Service-Oriented Architectures*. European Commission, EU project Digital Business Ecosystems (contract number 507953 [http://www.digital-ecosystem.org]). 2007. doi:10.1109/BIMNICS.2006.361817.

[22] Bundesministerium des Inneren. *Digitale Verwaltung 2020*. 2020.

[23] Bundesverband Informationswirtschaft Telekommunikation und Neue Medien e.V. *Vorschläge Zur Kommunalen Digitalen Transformation - Aufbau Eines Bundesweiten Kompetenzzentrums » Digitale Städte Und Regionen«*. 2018.

[24] Frank Buschmann, Kevlin Henney, und Douglas Schmidt. “Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing”. *Wiley*. John Wiley & Sons, 639. 2007. http://www.citeulike.org/group/1660/article/1686652.

[25] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, und Jean Vanderdonckt. “The CAMELEON Reference Framework. Components, 60.” 2002. http://giove.isti.cnr.it/projects/cameleon.html.

[26] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, und Jean Vanderdonckt. “A Unifying Reference Framework for Multi-Target User Interfaces”. *Interacting with Computers* 15 (3): 289–308. 2003. http://iwc.oxfordjournals.org/content/15/3/289.short.

[27] Cloves Carneiro, und Tim Schmelmer. *Microservices From Day One - Build Robust and Scalable Software from the Start*. APress. 2016. doi:10.1007/978-1-4842-1937-9.

[28] Stefano Stephano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, und Maristella Matera. *Designing Data-Intensive Web Applications*. Herausgegeben von Jim Gray. The Morgan. Morgan Kaufman. 2002.

[29] Tomas Cerny, Michael J. Donahoo, und Michal Trnka. “Contextual Understanding of Microservice Architecture”. *ACM SIGAPP Applied Computing Review* 17 (4): 29–45. 2018. doi:10.1145/3183628.3183631.

[30] Elizabeth Chang, und Martin West. “Digital Ecosystems A Next Generation of the Collaborative Environment”. In *iiWAS’2006 - The Eighth International Conference on Information Integration and Web-Based Applications Services, 4-6 December 2006, Yogyakarta, Indonesia*, 3–23. Austrian Computer Society. 2006.

[31] Kishore Channabasavaiah, Kerrie Holley, und Edward M Tuggle. “Migrating to a Service-Oriented Architecture”. *IBM DeveloperWorks*, Nummer April: 1–22. 2004. doi:10.1109/ICWS.2004.1314715.

[32] Mingming Cheng. “Sharing Economy: A Review and Agenda for Future Research”. *International Journal of Hospitality Management* 57. Elsevier Ltd: 60–70. 2016. doi:10.1016/j.ijhm.2016.06.003.

[33] L Cherbakov, G Galambos, R Harishankar, S Kalyana, und G Rackham. “Impact of Service Orientation at the Business Level”. *IBM Systems Journal* 44 (4): 653–668. 2005. doi:10.1147/sj.444.0653.

[34] Paul Chlebek. *User Interface-Orientierte Softwarearchitektur*. 1. Auflage. Vieweg & Sohn Verlag, Wiesbaden. 2006.

[35] Paul Chlebek. *Praxis Der User Interface-Entwicklung*. Vieweg+Teubner. 2011.

[36] Jacek Chmielewski, und Krzysztof Walczak. “Application Architectures for Smart Multi-Device Applications”. In *Proceedings of the Workshop on Multi-Device App Middleware - Multi-Device ’12*, 1–5. ACM Press, New York, New York, USA. 2012. doi:10.1145/2405172.2405177.

[37] Clayton M. Christensen. *The Innovators Dilemma: Warum Etablierte Unternehmen Den Wettbewerb Um Bahnbrechende Innovationen Verlieren*. Vahlen, München. 2011.

[38] Tony Clark, Paul Sammut, und James Willans. “Applied Metamodelling: A Foundation for Language Driven Development (Third Edition)”. *ArXiv E-Prints*. 2015. doi:10.1016/j.jbusres.2014.06.013.The.

[39] Paul Clements. “Software Product Lines”. *CROSSTALK The Journal of Defense Software Engineering*, Nummer February: 20–22. 1999.

[40] Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan van den Bergh, und Bert Creemers. *Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems*. Springer. 2003.

[41] Larry L Constantine, und Lucy A D Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. ACM Press/Addison-Wesley Publishing Co., New York. 1999. http://portal.acm.org/citation.cfm?id=301248.

[42] Thomas H. Cormen, Charles L. Leiserson, und Ronald L. Rivest. *Introduction to Algorithms (Third Edition)*. 2. The MIT Press, Cambridge, Massachusetts London, England. 2009.

[43] Joëlle Coutaz. “User Interface Plasticity: Model Driven Engineering to the Limit!” In *EICS ’10 Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 1–8. Eics. 2010. doi:10.1145/1822018.1822019.

[44] Richard Cyganiak, David Wood, und Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[45] Peter Dadam, und Manfred Reichert. “The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support”. *Computer Science - Research and Development* 23 (2): 81–97. Apr. 2009. doi:10.1007/s00450-009-0068-6.

[46] George Demarest, und Peter Heller. “Oracle Fusion Architecture and Oracle Fusion Applications”. Oracle Technical White Paper. Oracle Corporation. 2006.

[47] Gabriel Dermler, Michael Wasmund, Guido Grassel, Axel Spriestersbach, und Thomas Ziegert. “Flexible Pagination and Layouting for Device Independent Authoring”. TU Darmstadt. 2003. https://fileserver.tk.informatik.tu-darmstadt.de/Publications/2003/www2003.pdf.

[48] DGUV. *Softwareergonomie - DGUV Information 215-450*. August. Deutsche Gesetzliche Unfallversicherung e.V., Berlin. 2016. http://publikationen.dguv.de/dguv/pdf/10002/215-450.pdf.

[49] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, und Porfirio Tramontana. “Recovering Interaction Design Patterns in Web Applications”. *Ninth European Conference on Software Maintenance and Reengineering*, 366–374. 2005. doi:10.1109/CSMR.2005.47.

[50] Hai Dong, und Farookh Khadeer Hussain. “Focused Crawling for Automatic Service Discovery, Annotation, and Classification in Industrial Digital Ecosystems”. *IEEE Transactions on Industrial Electronics* 58 (6): 2106–2116. 2011. doi:10.1109/TIE.2010.2050754.

[51] Rainer Dorau. *Emotionales Interaktionsdesign*. Springer Heidelberg Dordrecht London New York. 2011. doi:10.1007/978-3-642-03101-4.

[52] Aline Dresch, Daniel Pacheco Lacerda, und José Antônio Valle Antunes Jr. *Design Science Research*. Springer International Publishing, Cham. 2015. doi:10.1007/978-3-319-07374-3.

[53] Micah Dubinko, Leigh Klotz, Roland Merrik, und T. Raman. “XForms 1.0 W3C Recommendation”. 2003.

[54] Roman Dumitru, Keller Uwe, Lausen Holger, Bruijn de Jos, Lara Rubén, Stollberg Michael, Polleres Axel, Feier Cristina, Bussler Cristoph, und Fensel Dieter. “Web Service Modeling Ontology”. 2005.

[55] Jürgen Dunkel, Andreas Eberhart, Carsten Kleiner, und Arne Koschel. *Systemarchitekturen Für Verteilte Anwendungen*. Carl Hanser Verlag, München. 2008.

[56] Thomas Erickson. “Working with Interface Metaphors ” In The Art of Human- Computer Interface Design”. In *Readings in Human-Computer Interaction: Toward the Year 2000*, 147–151. October. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA. 1995.

[57] Thomas Erl. *SOA: Principles of Service Design*. The Prenti. Prentice Hall. 2008.

[58] Dominik Ertl. “Semi-Automatic Multimodal User Interface Generation”. *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS ’09*. ACM Press, New York, New York, USA, 321. 2009. doi:10.1145/1570433.1570494.

[59] Jüergen Falb, Sevan Kavaldjian, Roman Popp, David Raneburger, Edin Arnautovic, und Hermann Kaindl. “Fully Automatic User Interface Generation from Discourse Models”. In *Proceedingsc of the 13th International Conference on Intelligent User Interfaces - IUI ’09*, 475. 2008. doi:10.1145/1502650.1502722.

[60] Jürgen Falb, Roman Popp, Thomas Röck, Helmut Jelinek, Edin Arnautovic, und Hermann Kaindl. “Fully-Automatic Generation of User Interfaces for Multiple Devices from a High-Level Model Based on Communicative Acts”. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, 1–10. 2007. doi:10.1109/HICSS.2007.236.

[61] David C. Fallside, und Priscilla Walmsley. “XML Schema Part 0: Primer Second Edition”. 2004. https://www.w3.org/TR/xmlschema-0/.

[62] Joel Farrell, und Holger Lausen. “Semantic Annotations for WSDL and XML Schema”. 2007. http://www.w3.org/TR/sawsdl/.

[63] Steve Faulkner, Arron Eicholz, Travis Leithead, Alex Danilo, und Sangwhan Moon. “HTML 5.2 Specification”. 2017. https://www.w3.org/TR/html52/.

[64] Irina Fedortsova. “Mastering FXML”. 2011. http://docs.oracle.com/javafx/2/.

[65] Pierfranco Ferronato. “Architecture for Digital Ecosystems, beyond Service Oriented Architecture (IEEE-DEST 2007)”. *2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, 660–665. 2007. doi:10.1109/DEST.2007.372047.

[66] Roy Thomas Fielding. “Architectural Styles and the Design of Network-Based Software Architectures”. Phdthesis, University of California, Irvine. 2000. doi:10.1.1.91.2433.

[67] Murielle Florins, Francisco Montero Simarro, Jean Vanderdonckt, und Benjamin Michotte. “Splitting Rules for Graceful Degradation of User Interfaces”. *Proceedings of the 11th International Conference on Intelligent User Interfaces - IUI ’06*. ACM Press, New York, New York, USA, 264. 2006. doi:10.1145/1111449.1111505.

[68] J. Foley, W. Kim, S. Kovacevic, und K. Murray. *UIDE – An Intelligent User Interface Design Environment*. Addison-Wesley, ACM Press. 1991.

[69] Jn Foster, Mb Greenwald, und Jt Moore. “Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem”. In *POPL ’05 Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Princi-*

ples of Programming Languages, 3:233–246. ACM New York, NY, USA. 2005. doi:10.1145/1232420.1232424.

[70] The Apache Software Foundation. "Apache Jena". https://jena.apache.org/.

[71] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional. 2010.

[72] Franz-Josef Fritz. "When Does a Web Service Become an Enterprise Service?" *SAP Insider* 5 (2). 2004.

[73] Krzysztof Gajos. "Automatically Generating Personalized User Interfaces". Phdthesis, University of Washington. 2008.

[74] Krzysztof Z. Gajos, Daniel S. Weld, und Jacob O. Wobbrock. "Automatically Generating Personalized User Interfaces with Supple". *Artificial Intelligence* 174 (12-13): 910–950. 2010. doi:10.1016/j.artint.2010.05.005.

[75] Raffaele Garofalo. *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern*. O'Reilly Media, Inc. 2011.

[76] Werner Gaulke, und Jürgen Ziegler. "Using Profiled Ontologies to Leverage Model Driven User Interface Generation". *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '15*, 254–259. 2015. doi:10.1145/2774225.2775070.

[77] A.S. Gibbons, und C.V. Bunderson. "Explore, Explain, Design". In *Encyclopedia of Social Measurement*, 927–938. 2003.

[78] T. Grandon Gill, und Alan R. Hevner. "A Fitness-Utility Model for Design Science Research". *ACM Transactions on Management Information Systems* 4 (2): 237–252. 2013. doi:10.1145/2499962.2499963.

[79] Jaime Gómez, und Cristina Cachero. "OO-H Method : Extending UML to Model Web Interfaces". *Information Modeling for Internet Applications*, 144–173. 2003.

[80] Google Inc. "Google Assistant". 2018. https://developers.google.com/assistant/sdk/.

[81] Google Inc. "Google Dialog Flow". 2018. https://dialogflow.com.

[82] Google Inc. "AngularJS". Mai 2018. Zugegriffen Mai 28. https://angularjs.org/.

[83] James Gosling, Bill Joy, Guy Steele, und Gilad Bracha. *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional. 2014. https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf.

[84] Volker Gruhn, Daniel Pieper, und Carsten Röttgers. *MDA*. Springer, Berlin Heidelberg. 2006.

[85] Josefina Guerrero-García, Juan Manuel González-Calleros, Jean Vanderdonckt, und Jaime Muñoz-Arteaga. “A Theoretical Survey of User Interface Description Languages: Preliminary Results”. *2009 Latin American Web Congress - Joint LA-WEB/CLIHC Conference*, 36–43. 2009. doi:10.1109/LA-WEB.2009.40.

[86] Chung Wei Hang, Anup K. Kalia, und Munindar P. Singh. “Behind the Curtain: Service Selection via Trust in Composite Services”. In *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 9–16. IEEE. 2012. doi:10.1109/ICWS.2012.96.

[87] Ute Hansen. *E-Government Im Kontext von Leistungsnetzwerken: 20 Kooperative Erfolgsfaktoren*. BookOnDemand - vabaduse. 2010.

[88] Qusay F. Hassan, Herausgeber. *Internet of Things A to Z: Technologies and Applications - Concepts and Perspectives*. June. Wiley-IEEE Press. 2018.

[89] Andreas M Heinecke. *Mensch-Computer-Interaktion: Basiswissen Für Entwickler Und Gestalter*. 2. Auflage. Springer. 2012.

[90] James Helms. *User Interface Markup Language (UIML) Specification V 4.0*. January. OASIS. 2008. http://docs.oasis-open.org/uiml/v4.0/cd01/uiml-4.0-cd01.pdf.

[91] Sebastian Herden, Jorge Marx Gómez, Claus Rautenstrauch, und André Zwanziger. *Software-Architekturen Für Das E-Business*. Springer, Berlin Heidelberg New York. 2006. https://sisis.rz.htw-berlin.de/inh2006/1235289.pdf.

[92] Roland Heuermann, Christian Bressem, und Matthias Tomenendal. *Digitalisierung in Bund , Ländern Und Gemeinden - IT-Organisation, Management Und Empfehlungen*. Springer-Verlag Berlin GmbH Deutschland. 2018. doi:10.1007/978-3-662-54098-5.

[93] Alan R Hevner, Salvatore T March, Jinsoo Park, und Sudha Ram. “Design Science in Information Systems Research”. *MIS Quaterly* 28 (1): 75–105. 2004.

[94] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, und Sebastian Rudolph. “OWL 2 Web Ontology Language Primer (Second Edition)”. Word Wide Web Consortium (W3C). 2012. https://www.w3.org/TR/owl2-primer/.

[95] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, und York Sure. *Semantic Web*. eXamen.press. Springer Berlin Heidelberg, Berlin, Heidelberg. 2008. doi:10.1007/978-3-540-33994-6.

[96] Michael Hitz. “Eine Multikanal-Architektur Für Frontendsysteme Und Deren Erweiterbarkeit Durch Variantenbildung”. In *Proceedings Software Engineering 2013, Workshopband, LNI 215*, 583–589. GI, Köllen Druck+Verlag GmbH, Bonn. 2013.

[97] Michael Hitz. “Konzept Zur Variantenbildung von Oberflächen in Einer Multikanal-Umgebung”. In *Lecture Notes in Informatics INFORMATIK 2013 - GI-Jahrestagung 2013, LNI 220*, 2664–2678. GI, Köllen Druck+Verlag GmbH, Bonn. 2013.

[98] Michael Hitz. *Interner Projektbericht Zum Themenfeld Mimesis . Ui*. 2014.

[99] Michael Hitz, und Thomas Kessel. “Using Application Ontologies for the Automatic Generation of User Interfaces for Dialog-Based Applications Data-Centric Description of Interview Applications”. In *Proceedings of CONFENIS 2016, International Conference on Research and Practical Issues of Enterprise Information Systems, LNBIP 268*, 16–31. Springer International Publishing AG. 2016. doi:10.1007/978-3-319-49944-4_2.

[100] Michael Hitz, Thomas Kessel, und Dennis Pfisterer. “Towards Sharable Application Ontologies for the Automatic Generation of UIs for Dialog Based Linked Data Applications”. SCITEPRESS – Science and Technology Publications, Lda. 2017.

[101] Michael Hitz, Mirjana Radonjic-simic, Julian Reichwald, und Dennis Pfisterer. “Generic UIs for Requesting Complex Products within Distributed Market Spaces in the Internet of Everything”. In *Proceedings of CD-ARES 2016, Availability, Reliability and Security in Information Systems, LNCS 9817*, herausgegeben von Francesco Buccafurri, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, und Edgar Weippl, 29–44. Springer International Publishing. 2016. doi:10.1007/978-3-319-45507-5 3.

[102] Masahiro Hori, Jérôme Euzenat, und Peter F. Patel-Schneider. “OWL Web Ontology Language XML Presentation Syntax”. 2003. http://www.w3.org/TR/owl-xmlsyntax/.

[103] Matthew Horridge, und Peter F. Patel-Schneider. “OWL 2 Web Ontology Language Manchester Syntax (Second Edition)”. 2009. https://www.w3.org/2007/OWL/wiki/ManchesterSyntax.

[104] International Organization for Standardization. *DIN EN ISO 9241-13:1998(en) - Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) — Part 13: User Guidance*. 1998.

[105] International Organization for Standardization. *ISO 13616-1:2007 - Financial Services - International Bank Account Number (IBAN) - Part 1: Structure of the IBAN*. ISO - International Organization for Standardization. 2007. https://www.iso.org/standard/41031.html.

[106] International Organization for Standardization. *International Standard ISO/IEC 11404: Information Technology — General- Purpose Datatypes (GPD) Technologies -ISO/IEC 11404:2007(E)*. Band 25021. 2007. doi:10.1109/IEEESTD.2015.7106438.

[107] International Organization for Standardization. *DIN EN ISO 9241-110:2008 - Ergonomie Der Mensch-System-Interaktion - Teil 110: Grundsätze Der Dialoggestaltung*. 2008.

[108] International Organization for Standardization. *DIN EN ISO 9241-143:2012 - Ergonomie Der Mensch-System-Interaktion - Teil 143: Formulardialoge (ISO 9241-*

143:2012); Deutsche Fassung EN ISO 9241-143:2012. ISO - International Organization for Standardization. 2012.

[109] International Organization for Standardization. *ISO 9362:2014 - Banking - - Banking Telecommunication Messages - Business Identifier Code (BIC)*. ISO - International Organization for Standardization. 2014. https://www.iso.org/standard/60390.html.

[110] International Organization for Standardization. *DIN EN ISO 9241-11:2018(en) - Ergonomics of Human-System Interaction - Part 11: Usability: Definitions and Concepts*. 2018.

[111] S Jablonski, und C Bussler. "Workflow Management Systems: Modelling Concepts, Architecture and Implementation". 1996. https://books.google.pt/books?id=hs0eAQAAIAAJ.

[112] Christian Janssen, Anette Weisbecker, und Jürgen Ziegler. "Generating User Interfaces from Data Models and Dialogue Net Specifications". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '93*, herausgegeben von S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, und T. White, 418–423. ACM Press New York. 1993. doi:10.1145/169059.169335.

[113] Sabina Jeschke, Christian Brecher, Houbing Song, und Danda B. Rawat. *Industrial Internet of Things - Cybermanufacturing Systems*. Springer International Publishing Switzerland. 2017. doi:10.1007/978-3-319-42559-7.

[114] Nicolai Josuttis. *SOA in Practice - the Art of Distributed Design*. O'Reilly Media, Sebastopol. 2007.

[115] Deepali Khushraj, und Ora Lassila. "Ontological Approach to Generating Personalized User Interfaces for Web Services". *The Semantic Web–ISWC 2005*, 916–927. 2005. doi:10.1007/11574620_65.

[116] N Kiyavitskaya, und N Zeni. "A Lightweight Approach to Semantic Tagging." *WWW Workshop on* . 2004. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-105/Semantic%20tagging{_}NYC.pdf.

[117] Melanie Knorsch, Theresa Schmidt, Therese Ho, Laura Sieweke, und Julia Wahle. "Typisierung von Anwendungen in Online-Portalen". Studienarbeit, Duale Hochschule Baden-Württemberg, Stuttgart. 2015.

[118] Donald E. Knuth. *The Art of Computer Programming - Volume 1 - Fundamental Algorithms*. Addison Wesley Longman. 1997.

[119] Nora Koch, und Sergej Kozuruba. "Requirements Models as First Class Entities in Model-Driven Web Engineering". In *Current Trends in Web Engineering. ICWE 2012. Lecture Notes in Computer Science, Vol 7703*, herausgegeben von Wimmer M. Gross-

niklaus M., 158–169. Springer, Berlin, Heidelberg. 2012. doi:10.1007/978-3-642-35623-0_16.

[120] Nora Koch, und Andreas Kraus. “Towards a Common Metamodel for the Design of Web Applications”. In *Proceedings of the 3rd Intl Conference on Web Engineering (ICWE 2003), LNCE 2722*. Springer. 2003.

[121] Nora Koch, Santiago Meliá-beigbeder, Nathalie Moreno-vergara, Vicente Pelechano-ferragud, und Fernando Sánchez-figueroa. “Model-Driven Web Engineering”. *UP-Grade* IX (2): 40–45. 2008.

[122] N Koppenhagen, O Gaß, und B Müller. “Design Science Research in Action - Anatomy of Success Critical Activities for Rigor and Relevance”. *Proceedings of the 20th European Conference on Information Systems (ECIS 2012)*. 2012. doi:10.5445/IR/1000055012.

[123] Dirk Krafzig, Karl Banke, und Dirk Slama. *Enterprise SOA – Best Practices Für Serviceorientierte Architekturen – Einführung, Umsetzung, Praxis*. Coad Serie. Prentice Hall. 2005.

[124] Masaaki Kurosu, Herausgeber. *Human-Computer Interaction: Interaction Modalities and Techniques*. Lecture No. Springer. 2013.

[125] Ulrich Küster, Birgitta König-Ries, Michael Klein, und Mirco Stern. “DIANE: A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding, and Invocation on the Web”. *International Journal of Electronic Commerce* 12 (2): 41–68. 2007. doi:10.2753/JEC1086-4415120202.

[126] Marc M Lankhorst, und Paul H W M Oude Luttighuis. “Enterprise Architecture Patterns for Multichannel Management 2 Characteristics of Multichannel Management”. In *Software Engineering 2009 - Workshopband, Fachtagung Des GI-Fachbereichs Softwaretechnik Kaiserslautern*, 31–42. Telematica Instituut PO Box 589 7500 AN Enschede The Netherlands. 2009. http://subs.emis.de/LNI/Proceedings/Proceedings150/article4836.html.

[127] Latitude. “’The New Sharing Economy’, a Study by Latitude in Collaboration with Shareable Magazine”. *Shareable Magazine*. 2010. http://latdsurvey.net/pdf/Sharing.pdf.

[128] Patrick Lay, und Stefan Lüttringhaus-Kappel. “Transforming XML Schemas into Java Swing GUIs”. *GI Jahrestagung*, 271–276. 2004. http://subs.emis.de/LNI/Proceedings/Proceedings50/article3155.html.

[129] Daniel Liebhart. *SOA Goes Real*. Carl Hanser Verlag, München. 2007.

[130] Quentin Limbourg. “USIXML: A User Interface Description Language Supporting Multiple Levels of Independence.” In *ICWE Workshops*, herausgegeben von Maristella Matera und Sara Comai, 325–338. Rinton Press. 2004. http://www.

pst.informatik.uni-muenchen.de/{~}baumeist/icwe/ws/ws4/DIWE2004-Limbourg.pdf.

[131] Claudia Linnhoff-Popien, Ralf Schneider, und Michael Zaddach, Herausgeber. *Digital Marketplaces Unleashed*. Springer, Berlin, Heidelberg. 2018. doi:10.1007/978-3-662-49275-8_1.

[132] Martin Lippert, Henning Wolf, und Heinz Züllighoven. "Domain Services for Multichannel Application Software". In *Proceedings of the Hawaii International Conference On System Sciences (HICSS)*. 2001.

[133] Ben Liu, Hejie Chen, und Wei He. "Deriving User Interface from Ontologies: A Model-Based Approach". *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI* 2005: 254–259. 2005. doi:10.1109/ICTAI.2005.55.

[134] J Ludewig, und H Lichter. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. Dpunkt.Verlag GmbH, Heidelberg. 2013.

[135] Claude R. Martin, und David A. Horne. "Restructuring towards a Service Orientation: The Strategic Challenges". *International Journal of Service Industry Management* 3 (1): 25–38. 1992. doi:10.1108/EUM0000000002809.

[136] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, u. a. "OWL-S : Semantic Markup for Web Services". 2004. https://www.w3.org/Submission/OWL-S/.

[137] Scott McGlashan, Daniel C. Burnett, Jerry Carter, Peter Danielsen, Jim Ferrans, Andrew Hunt, Bruce Lucas, Brad Porter, Ken Rehor, und Steph Tryphonas. "Voice Extensible Markup Language (VoiceXML) Version 2.0". 2004. http://www.w3.org/TR/voicexml20/.

[138] Ben de Meester, Anastasia Dimou, Ruben Verborgh, und Erik Mannens. "An Ontology to Semantically Declare and Describe Functions". *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9989 LNCS: 46–49. 2016. doi:10.1007/978-3-319-47602-5_10.

[139] Gerrit Meixner, und Christian Müller. *Automotive User Interfaces - Creating Interactive Experiences in the Car*. Springer International Publishing AG. 2017. doi:10.1007/978-3-319-49448-7.

[140] Gerrit Meixner, Fabio Paternò, und Jean Vanderdonckt. "Past, Present, and Future of Model-Based User Interface Development". *I-Com Zeitschrift Für Interaktive Und Kooperative Medien* 10 (3): 2–11. 2011. doi:10.1524/icom.2011.0026.

[141] Stephen J. Mellor, Anthony N. Clark, und Takao Futagami. "Model-Driven Development". *IEEE Software* 20 (5): 14–18. 2003. doi:10.1109/MS.2003.1231145.

[142] Stephen J Mellor, Kendall Scott, Axel Uhl, und Dirk Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley Professional. 2004.

[143] D Merril. "Mashups: The New Breed of Web App - an Introduction to Mashups". *IBM Developer Works* 2009: 1–13. 2006. https://www.ibm.com/developerworks/library/x-mashups/.

[144] Joaquin Miller, und Jishnu Mukerji. *MDA Guide Version 1.0.1*. June. Band 234. Object Management Group (OMG). 2003. doi:10.1074/jbc.M312687200.

[145] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valérie Issarny, und Yolande Berbers. "EASY: Efficient semAntic Service discoverY in Pervasive Computing Environments with QoS and Context Support". *Journal of Systems and Software* 81 (5): 785–808. 2008. doi:10.1016/j.jss.2007.07.030.

[146] James F. Moore. *The Death of Competition: Leadership and Strategy in the Age of Business Ecosystems*. Harper Paperbacks. 1997.

[147] Martin Moosbauer. "Analyse Der Verwendbarkeit von XSD Zur Beschreibung von Anwendungsfrontends in Mimesis". Projektarbeit, Duale Hochschule Baden-Württemberg, Stuttgart, Stuttgart. 2016.

[148] G. Mori, F. Paterno, und C. Santoro. "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions". *IEEE Transactions on Software Engineering* 30 (8): 507–520. Aug. 2004. doi:10.1109/TSE.2004.40.

[149] Mozilla. "Mozilla. XUL. Aufgerufen von: https://developer.mozilla.org/en/XUL". 2014. https://developer.mozilla.org/en/XUL.

[150] Jeffrey Nichols. "Automatically Generating High-Quality User Interfaces for Appliances". Phdthesis, Carnegie Mellon University. 2006.

[151] Jeffrey Nichols, und Brad a. Myers. "Creating a Lightweight User Interface Description Language: An Overview of the Personal Universal Controller Project". *ACM Transactions on Computer-Human Interaction* 16 (4). Nov. 2009. doi:10.1145/1614390.1614392.

[152] Jakob Nielsen. *Designing User Interfaces for International Use*. Emerald Group Publishing Limited. 1990.

[153] Erik G. Nilsson. "Design Patterns for User Interface for Mobile Applications". *Advances in Engineering Software* 40 (12). Elsevier Ltd: 1318–1328. 2009. doi:10.1016/j.advengsoft.2009.01.017.

[154] Linda M Northrop, Paul C Clements, With Felix Bachmann, John Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, und Lawrence Jones. *A Framework for Software Product Line Practice, Version 5 . 0*. 2016.

[155] OASIS Open. *Reference Model for Service Oriented Architecture. OASIS Standard, 12 October 2006*. October. 2006. https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html.

[156] OASIS Open. *Reference Architecture Foundation for Service Oriented Architecture Version 1.0. 04 December 2012*. 2012. http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html.

[157] Object Management Group. *Unifed Modeling Language (UML), Version 2.5*. 2015. http://www.omg.org/spec/UML/2.5/.

[158] Oracle Corporation. "JSR 341 Expression Language Specification Version 3.0". Oracle Corporation. 2013.

[159] Eyal Oren, Knud Hinnerk Möller, Simon Scerri, Siegfried Handschuh, und Michael Sintek. "What Are Semantic Annotations?" *Relatório Técnico. DERI Galway* 9: 14. 2006.

[160] Sharon Oviatt. "Multimodal Interfaces". In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, herausgegeben von Andrew Sears und Julie A. Jacko, 3rd Auflage, 403ff, Kapitel 18. CRC Press, Boca Raton, FL, USA. 2012.

[161] "OWLAPI". http://owlcs.github.io/owlapi/.

[162] Dariusz Parys, und Jürgen Mauerer. "Web Services Standards: SOAP, UDDI Und WSDL". 2004. https://msdn.microsoft.com/de-de/library/bb979447.aspx.

[163] Fabio Paterno, Carmen Santoro, und Lucio Davide Spano. "Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment". *ACM Transactions on Computer-Human Interaction* 16 (4). Nov. 2009. doi:10.1145/1614390.1614394.

[164] Sanjay Patni. *Pro RESTful APIs*. Apress, Berkeley, CA. 2017. doi:10.1007/978-1-4842-2665-0.

[165] Ken Peffers, Marcus Rothenberger, Tuure Tuunanen, und Reza Vaezi. "Design Science Research Evaluation". *Design Science Research in Information Systems. Advances in Theory and Practice*, 398–410. 2012. doi:10.1007/978-3-642-29863-9_29.

[166] Matthias Peissner. *Entwurfsmusterbasierter Ansatz Für Adaptive Benutzungsschnittstellen Zur Überwindung von Nutzungsbarrieren*. July. 2015. doi:10.18419/opus-6878.

[167] Matthias Peißner, Doris Janssen, und Thomas Sellner. "MyUI Individualization Patterns for Accessible and Adaptive User Interfaces". In *SMART 2012—The First International Conference on Smart Systems, Devices and Technologies*, 25–30. c. 2012.

[168] Dennis Pfisterer, Mirjana Radonjic-Simic, und Julian Reichwald. *Business Model Design and Architecture for the Internet of Everything*. Band 5. Apr. 2016. doi:10.3390/jsan5020007.

[169] Georg Pietrek, und Jens Trompeter, Herausgeber. *Modellgetriebene Softwareentwicklung: MDA Und MDSD in Der Praxis*. Entwickler.Press, Frankfurt am Main. 2007.

[170] Andreas Pleuss, Goetz Botterweck, und Deepak Dhungana. "Integrating Automated Product Derivation and Individual User Interface Design". In *Proceedings Vamos'10*, herausgegeben von David; Benavides, Don Batory, und Paul Grünbacher, 69–77. 2010.

[171] Andreas Pleuss, Benedikt Hauptmann, und Deepak Dhungana. "User Interface Engineering for Software Product Lines: The Dilemma between Automation and Usability". In *EICS "12 Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing systemsEICS"12*, 25–34. 2012. doi:10.1145/2305484.2305491.

[172] Andreas Pleuss, Benedikt Hauptmann, Markus Keunecke, und Goetz Botterweck. "A Case Study on Variability in User Interfaces". In *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -Volume 1*, 6. 2012. doi:10.1145/2362536.2362542.

[173] Andreas Pleuss, Stefan Wollny, und Goetz Botterweck. "Model-Driven Development and Evolution of Customized User Interfaces". In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '13*, 13. ACM Press, New York, New York, USA. 2013. doi:10.1145/2494603.2480298.

[174] K. Pohl, G. Böckle, und F. Van Der Linden. *Software Product Line Engineering. Foundations, Principles, and Techniques*. Band 49. 12. 2005. doi:10.1007/3-540-28901-1.

[175] Roman Popp, Jürgen Falb, Edin Arnautovic, Hermann Kaindl, Sevan Kavaldjian, Dominik Ertl, Helmut Horacek, und Cristian Bogdan. "Automatic Generation of the Behavior of a User Interface from a High-Level Discourse Model". In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*. 2009. doi:10.1109/HICSS.2009.84.

[176] Roman Popp, Jürgen Falb, David Raneburger, und Hermann Kaindl. "A Transformation Engine for Model-Driven UI Generation". *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '12*, 281. 2012. doi:10.1145/2305484.2305532.

[177] Bernhard Preim, und Raimund Dachselt. *Interaktive Sytsteme - Band 1*. eXamen.pre. Springer Vieweg, Berlin Heidelberg. 2010.

[178] Jan Pries-Heje, Richard L. Baskerville, und John R. Venable. "Strategies for Design Science Research Evaluation". *European Conference on Information Systems (ECIS)* Paper 87: 1–13. 2008. doi:10.1177/1933719108329095.

[179] Eric Prud'hommeaux, und Gavin Carothers. "RDF 1.1 Turtle: Terse RDF Triple Language". 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[180] A. Puerta, und J. Eisenstein. "Towards a General Computational Framework for Model-Based Interface Development Systems". *Knowledge-Based Systems* 12 (8): 433–442. Dez. 1999. doi:10.1016/S0950-7051(99)00037-4.

[181] Angel Puerta. *The MECANO Project : Comprehensive and Integrated Support for Model-Based Interface Development*. Namur University Press. 1996. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.92.6384.

[182] Angel R. Puerta, Henrik Eriksson, John H. Gennari, und Mark A. Musen. "Beyond Data Models for Automated User Interface Generation". In *Proceedings British HCI'94*. 1994. doi:10.1017/CBO9780511600821.027.

[183] Mirjana Radonjic-Simic, und Dennis Pfisterer. "A Decentralized Business Ecosystem Model for Complex Products". In *Digital Business*, herausgegeben von Tavana M., Popentiu-Vlădicescu F.}, Qiao F. Patnaik S. Yang XS., 23–52. Springer International Publishing. 2019. doi:10.1007/978-3-319-93940-7_2.

[184] Mirjana Radonjic-Simic, und Dennis Pfisterer. "Reference Model and Architecture for the Post-Platform Economy". In *Economics of Digital Transformation*, 465. University of Rijeka, Faculty of Economics and Business. 2019.

[185] Mirjana Radonjic-Simic, Frank Wolff, und Dennis Pfisterer. "Analyzing a Business Ecosystem for Complex Consumer Services". In *ICServ2017 - The 5th International Conference on Serviceology: Lecture Notes in Computer Science*, 10371 LNCS:46–52. Springer International Publishing, 2017. 2017. doi:10.1007/978-3-319-61240-9_-5.

[186] David Raneburger. "Interactive Model Driven Graphical User Interface Generation". *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '10*. ACM Press, New York, New York, USA, 321. 2010. doi:10.1145/1822018.1822071.

[187] David Raneburger, Hermann Kaindl, und Roman Popp. "Model Transformation Rules for Customization of Multi-Device Graphical User Interfaces". *EICS 2015*, 100–109. 2015.

[188] David Raneburger, Roman Popp, Hermann Kaindl, Jürgen Falb, und Dominik Ertl. "Automated Generation of Device-Specific WIMP UIs: Weaving of Structural and Behavioral Models". *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 41–46. 2011. doi:10.1145/1996461.1996492.

[189] David Raneburger, Roman Popp, und Jean Vanderdonckt. "An Automated Layout Approach for Model-Driven WIMP-UI Generation". *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '12*, 91. 2012. doi:10.1145/2305484.2305501.

[190] Trygve Reenskaug. *Models-Views-Controllers - Technical Note*. Xerox PARC. 1979.

[191] Adam Richardson. “Using Customer Journey Maps to Improve Customer Experience”. *Harward Business Review*. 2010.

[192] A.G.L Romme. “Making a Difference: Organization as Design”. *Organization Science* 14 (5): 558–573. 2003.

[193] Michael Rosen, Boris Lublinsky, Kevin T. Smith, und Marc J. Balcer. *Applied SOA: Service-Oriented Architecture and Design Strategies*. John Wiley & Sons. 2012.

[194] Christiane Rudlof. “Objekte Im User Interface - Probleme Ihrer Benennung”. Dissertation, Universität Bremen. 2009. http://elib.suub.uni-bremen.de/diss/docs/00011414.pdf.

[195] Toni Ruokolainen. “A Model-Driven Approach to Service Ecosystem Engineering”. Phdthesis, University of Helsinki, Finland. 2013.

[196] Chris Rupp, Stefan Queins, und die SOPHISTen. *UML 2 Glasklar*. 4. Auflage. Carl Hanser Verlag GmbH & Co. KG, München. Apr. 2012. doi:10.3139/9783446431973.

[197] Anila Sahar, Butt Armin, Haller Shepherd, und Liu Lexing. “ActiveRaUL : Automatically Generated Web Interfaces for Creating RDF Data”. *Proposal - Semantic Web, IOS Press* 0. 2013.

[198] Jan W Schemm, Christine Legner, und Rudolf Zurmühlen. “Evolution of Process Portals to Multi-Channel Architectures – A Service-Oriented Approach at ETA SA”. In , 1–19. 2006.

[199] Thomas Schlegel. *Multi-Touch*. Herausgegeben von Thomas Schlegel. Springer-Verlag Berlin Heidelberg. 2014. doi:10.1007/978-3-642-36113-5.

[200] Juliet B. Schor, und Connor J. Fitzmaurice. “Collaborating and Connecting: The Emergence of the Sharing Economy”. In *Handbook of Research on Sustainable Consumption*, herausgegeben von Lucia Reisch und John Thogersen, Social and, 410–425. Edward Elgar Publishing Ltd., Cheltenham, UK. 2015. doi:10.4337/9781783471270.00039.

[201] Gerti Schrefl, und Michael Kappel. *Objektorientierte Informationssysteme - Konzepte, Darstellungsmittel, Methoden*. Springers . Springer, Vienna. 1996. doi:10.1007/978-3-7091-9469-0.

[202] W. Roy Schulte, und Yefim V. Natis. *Service Oriented Architectures, Part 1. SSA Research Note SPA-401-068*. Gartner, Inc. 1996.

[203] W. Roy Schulte, und Yefim V. Natis. *“Service Oriented” Architectures, Part 2. SSA Research Note SPA-401-069*. Gartner, Inc. 1996.

[204] Wieland Schwinger, und Nora Koch. “Modeling Web Applications”, 39–64. 2006.

[205] Bill Scott, und Theresa Neil. *Designing Web Interfaces - Principles and Patterns for Rich Interactions*. 1st Editio. O’Reilly, Sebastopol. 2009.

[206] Robert Sedgewick. *Algorithmen in Java*. 3. Auflage. Pearson Education. 2003.

[207] Ahmed Seffah, und Homa Javahery. *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*. John Wiley & Sons, Ltd. 2005. doi:10.1002/0470091703.

[208] Maung K Sein, Ola Henfridsson, und Matti Rossi. “Action Design Research”. *MIS Quarterly* 35 (2): 1–20. 2011.

[209] “Sesame Framework”. https://www.w3.org/2001/sw/wiki/Sesame.

[210] Ben Shneiderman, und Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 4th Ed. Pearson Education. 2005.

[211] Paulo Silva. “User Interface Declarative Models and Development Environments: A Survey”. In *Interactive Systems Design, Specification, and Verification, DSV-IS 2000. Lecture Notes in Computer Science*, herausgegeben von P. Palanque und F. Paternò, 1946:207–226. 1. Springer, Berlin, Heidelberg. 2001. doi:10.1007/3-540-44675-3_13.

[212] Herbert Alexander Simon. *The Sciences of the Artificial*. Third Edit. MIT Press, USA. 1996.

[213] Manu Sporny, Gregg Kellogg, und Markus Lanthaler. “JSON-LD 1.0”. 2014.

[214] David Sprott, und Lawrence Wilkes. “Understanding Service-Oriented Architecture”. 2004. http://msdn.microsoft.com/en-us/library/aa480021.aspx.

[215] Thomas Stahl, Markus Völter, S Efftinge, A Haase, J Bettin, S Helsen, und M Kunz. *Modellgetriebene Softwareentwicklung*. dpunkt.verlag. 2007.

[216] Stanford Center for Biomedical Informatics Research. “Protég&eacute”; 2018. https://protege.stanford.edu.

[217] Torsten Stapelkamp. *Screen- Und Interfacedesign: Gestaltung Und Usability Für Hard- Und Software*. Springer Berlin Heidelberg. 2007.

[218] Pedro Szekely. “Retrospective and Challenges for Model-Based Interface Development”. *Design, Specification and Verification of Interactive Systems ’96*, 1–27. 1996. doi:10.1007/978-3-7091-7491-3_1.

[219] P. Szekely, P. Luo, und R. Neches. “Facilitating the Exploration of Inter- Face Design Alternatives: The HUMANOID Model of Interface Design”. In *Proceedings of SIGCHI’92*, 507–515. 1992.

[220] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, und E. Salcher. “Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach”. In *Engineering for Human-Computer Interaction*, 120–150. Chapman & Hall, London, UK. 1996.

[221] Andrew S. Tanenbaum, und Maarten van Steen. *Verteilte Systeme*. Pearson St. Pearson. 2008.

[222] Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O’Reilly Media. 2006.

[223] Vi Tran, Jean Vanderdonckt, Ricardo Tesoriero, und François Beuvens. “Systematic Generation of Abstract User Interfaces”. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS ’12*, 101–110. ACM Press, New York, New York, USA. 2012. doi:10.1145/2305484.2305502.

[224] Russ Unger, und Carolyn Chandler. *A Project Guide to UX-Design For User Experience Designers*. 1st Editio. New Riders, Berkeley. 2009.

[225] Vijay Vaishnavi, Bill Kuechler, und Stacie Petter. “Design Science Research in Information Systems”. http://desrist.org/design-research-in-information-systems/ .

[226] A Vallecillo, N Koch, C Cachero, S Comai, P Fraternali, I Garrigós, J Gómez, u. a. “MDWEnet: A Practical Approach to Achieve Interoperability of Model-Driven Web Engineering Methods”. In *Proceedings of the 3rd International Workshop on Model-Driven Web Engineering MDWE 2007, Como, Italy,*. May 2014. 2007.

[227] Jean Vanderdonckt. “Knowledge-Based Systems for Automated User Interface Generation: The TRIDENT Experience”. 1995.

[228] Jean M. Vanderdonckt, und François Bodart. “Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection”. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI ’93*, Nummer December: 424–429. 1993. doi:10.1145/169059.169340.

[229] J.R. Venable. “The Role of Theory and Theorising in Design Science Research.” *DESRIST* 24–25: 1–18. 2006.

[230] Markus Völter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, und Guido Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. 2013. http://dslbook.org.

[231] Martijn Van Welie, und Hallvard Trætteberg. “Interaction Patterns in User Interfaces”. *7th. Pattern Languages of Programs Conference 2000*, 13–16. 2000.

[232] Klaus Zeppenfeld, und Patrick Finger. *SOA Und WebServices*. Informatik Im Fokus. Springer, Berlin, Heidelberg. 2009. doi:10.1007/978-3-540-76991-0.