Universität Potsdam
Hasso-Plattner-Institut für Softwaresystemtechnik
Fachgebiet Systemanalyse und Modellierung

Virtual Prototypes for the Model-Based Elicitation and Validation of Collaborative Scenarios

Gregor Berg

June 2013

Dissertation
zur Erlangung des akademischen Grades
"Doktor der Ingenieurwissenschaften" (Dr. Ing.)
in der Wissenschaftsdisziplin
"Systemanalyse und Modellierung"

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2014/6972/
URN urn:nbn:de:kobv:517-opus-69729
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-69729

Abstract

Requirements engineers have to elicit, document, and validate how stakeholders act and interact to achieve their common goals in collaborative scenarios. Only after gathering all information concerning *who* interacts with *whom* to do *what* and *why*, can a software system be designed and realized which supports the stakeholders to do their work. To capture and structure requirements of different (groups of) stakeholders, scenario-based approaches have been widely used and investigated. Still, the elicitation and validation of requirements covering collaborative scenarios remains complicated, since the required information is highly intertwined, fragmented, and distributed over several stakeholders. Hence, it can only be elicited and validated collaboratively. In times of globally distributed companies, scheduling and conducting workshops with groups of stakeholders is usually not feasible due to budget and time constraints. Talking to individual stakeholders, on the other hand, is feasible but leads to fragmented and incomplete stakeholder scenarios. Going back and forth between different individual stakeholders to resolve this fragmentation and explore uncovered alternatives is an error-prone, time-consuming, and expensive task for the requirements engineers. While formal modeling methods can be employed to automatically check and ensure consistency of stakeholder scenarios, such methods introduce additional overhead since their formal notations have to be explained in each interaction between stakeholders and requirements engineers. Tangible prototypes as they are used in other disciplines such as design, on the other hand, allow designers to feasibly validate and iterate concepts and requirements with stakeholders.

This thesis proposes a model-based approach for *prototyping* formal behavioral specifications of stakeholders who are involved in collaborative scenarios. By simulating and animating such specifications in a remote domain-specific visualization, stakeholders can *experience* and validate the scenarios captured so far, i.e., how other stakeholders act and react. This interactive scenario simulation is referred to as a model-based virtual prototype. Moreover, through observing how stakeholders interact with a *virtual* prototype of their collaborative scenarios, formal behavioral specifications can be automatically derived which complete the otherwise fragmented scenarios. This, in turn, enables requirements engineers to elicit and validate collaborative scenarios in individual stakeholder sessions – *decoupled*, since stakeholders can participate remotely and are not forced to be available for a joint session at the same time. This thesis discusses and evaluates the feasibility, understandability, and modifiability of model-based virtual prototypes. Similarly to how physical prototypes are perceived, the presented approach brings behavioral models closer to being tangible for stakeholders and, moreover, combines the advantages of joint stakeholder sessions and decoupled sessions.

Zusammenfassung

Anforderungsingenieure erheben, dokumentieren und validieren wie Bedarfsträger in einzelnen und gemeinsamen Aktivitäten die Ziele ihrer kollaborativen Szenarios erreichen. Auf Grundlage von Angaben darüber, *wer warum* mit *wem* zusammen *was* erledigt, kann anschließend ein Softwaresystem spezifiziert und umgesetzt werden, welches die Bedarfsträger bei der Durchführung ihrer Abläufe unterstützt. Um Anforderungen verschiedener (Gruppen von) Bedarfsträger zu erfassen und zu strukturieren, werden szenariobasierte Ansätze genutzt und erforscht. Die Erhebung und Validierung von Anforderungen, die kollaborative Szenarios abdecken, ist dennoch kompliziert, da derartige Informationen hochgradig verknüpft, fragmentiert und über mehrere Bedarfsträger verteilt sind, wodurch sie nur in Gruppensitzungen effizient erhoben und validiert werden können. In Zeiten global verteilter Firmen ist die Planung und Durchführung solcher Workshops mit Gruppen von Bedarfsträgern nur selten praktikabel. Mit einzelnen Bedarfsträgern zu sprechen ist hingegen oft realisierbar, führt aber zu fragmentierten, unvollständigen Szenariobeschreibungen. Durch eine Vielzahl von Einzelgesprächen mit wechselnden Bedarfsträgern kann diese Fragmentierung aufgelöst werden – dies ist aber eine fehleranfällige und zeitaufwändige Aufgabe. Zwar bieten formale Modellierungsmethoden z.B. automatische Konsistenzchecks für Szenarios, doch führen derartige Methoden zu Mehraufwand in allen Gesprächen mit Bedarfsträgern, da diesen die verwendeten formalen Notationen jedes Mal erläutert werden müssen. Handfeste Prototypen, wie sie in anderen Disziplinen eingesetzt werden, ermöglichen es Designern, ihre Konzepte und erhobenen Anforderungen ohne viel Aufwand mit Bedarfsträgern zu validieren und zu iterieren.

In dieser Dissertation wird ein modellbasierter Generierungsansatz vorgeschlagen, der kollaborative Szenarios *prototypisch* auf Grundlage von formalen Verhaltensmodellen für die beteiligten Bedarfsträger darstellt. Durch die Simulation dieses Verhaltens und dessen Animation innerhalb einer webbasierten, domänenspezifischen Visualisierung, können Bedarfsträger diese Modelle *erleben* und die bisher erfassten Szenarios validieren. Eine derartige interaktive Szenariosimulation wird als modellbasierter *virtueller* Prototyp bezeichnet. Basierend auf den Interaktionen zwischen Bedarfsträgern und einem virtuellen Prototypen ihrer Szenarios können zudem formale Verhaltensspezifikationen automatisch abgeleitet werden, die wiederum die fragmentierten kollaborativen Szenarios vervollständigen. Dies ermöglicht es den Anforderungsingenieuren, die kollaborativen Szenarios in individuellen Sitzungen mit einzelnen Bedarfsträgern zu erheben und zu validieren – *entkoppelt* voneinander, da Bedarfsträger webbasiert teilnehmen können und dabei nicht darauf angewiesen sind, dass andere Bedarfsträger ebenfalls in der gleichen Sitzung teilnehmen. Diese Dissertation diskutiert und evaluiert die Machbarkeit, Verständlichkeit sowie die Änderbarkeit der modellbasierten virtuellen Prototypen. Auf die gleiche Art wie physikalische Prototypen wahrgenommen werden, erlaubt es der vorgestellte Ansatz, Verhaltensmodelle für Bedarfsträger erlebbar zu machen und so die Vorteile von Gruppensitzungen mit denen entkoppelter Sitzungen zu verbinden.

Acknowledgment

First and foremost, I'd like to express my gratitude to my supervisor Prof. Dr. Holger Giese at the System Analysis and Modeling group who provided me with the opportunity of pursuing my research. Secondly, I am grateful for the generous grant provided by the *HPI-Stanford Design Thinking Research Program*, as well as the collaborations and discussions within this research community, especially Thomas Beyhl, Jonathan Edelman, Alexander Lübbe, and Andreas Seibel. Concerning the identification of potential topics for this thesis, I had the pleasure of collaborating with Jörn Hartwig and Alexander Renneberg at D-LABS GmbH.

Concerning the implementation of the concepts presented in this thesis, I am grateful for the time, energy, and commitment of the following students (in alphabetical order): Daniel Eichler, Stefan Kleff, Alexander Lüders, Nico Rehwaldt, Stefan Richter, Henrik Steudel, and Ralf Teusner. For the supervision of some of the experiments, the help of Chriss Kühnl and Andreas Seibel was also greatly appreciated. Moreover, I would like to thank Regina Hebig, Stephan Hildebrandt, Christian Krause, and Ralf Teusner for providing valuable feedback on early drafts of this thesis and all my friends and colleagues at the System Analysis and Modeling group for their support and fruitful discussions. Furthermore, I am most grateful to Jessica Thomas for proof-reading my thesis.

Last but not least, I'm very grateful for the support and patience of my wife Denise who provided the all-embracing remainder of what enabled me to finish this thesis.

Contents

1. Introduction

In recent decades, software systems have become more complex, ubiquitous and interconnected. Nowadays, as digitalization progresses and computational power steadily increases, prospective end users of planned software systems make greater demands concerning the capabilities and the usability of such systems. Consequently, their complexity increases – as does the effort necessary to capture and define the needs of the system's end users as well as all other persons or organizations who "influence a system's requirements" or who are "impacted by that system", i.e. *stakeholders* as defined by Glinz and Wieringa [GW07]. Hence, stakeholders who need to be interviewed do not only include prospective end users of the planned system, but rather everybody in its environment [Ale05]. As modern software systems such as the ones developed for insurances or online marketplaces impact more people every day, this elicitation grows more complex to ensure that software engineers "get the right design as well as get the design right" [Bux07].

It is up to requirements engineers to elicit, document, and validate such needs, constraints and, more broadly, *requirements*. The IEEE's[1] *Standard Glossary of Software Engineering Terminology* [IEE90] defines a requirement as a *condition* or *capability* that is either needed by a user for solving a problem or achieving an objective, or that must be met or possessed by a system to satisfy a contract, standard, or specification. To gather requirements for a software system, requirements engineers and stakeholders interact in such a way that the stakeholders' domain knowledge is ascertained along with requirements based on specific domain properties [Jac00]. The interaction between stakeholder and requirements engineer was suitably summarized by Maiden and Alexander who pointed out that requirements engineers "get people to tell the stories of what their systems are meant to do, so they build the right thing" [AM04]. Still, building the right thing also includes the identification of all stakeholders, the elicitation of their requirements concerning the projected software system and the validation, which has to establish that all stakeholders have been correctly understood. Then, these requirements can be used to establish the initial vision, i.e. the idea of why the software system is needed, in context [JP93].

Requirements engineers are usually no experts in the stakeholders' domain – a fact that is potentially even helpful as it requires them to question underlying assumptions and tacit knowledge which otherwise would not be covered in the requirements they produce [Ber95, Ber02]. Nevertheless, this also implies that a requirements engineer's

[1] The Institute of Electrical and Electronics Engineers, `http://www.ieee.org/` (accessed June 2013)

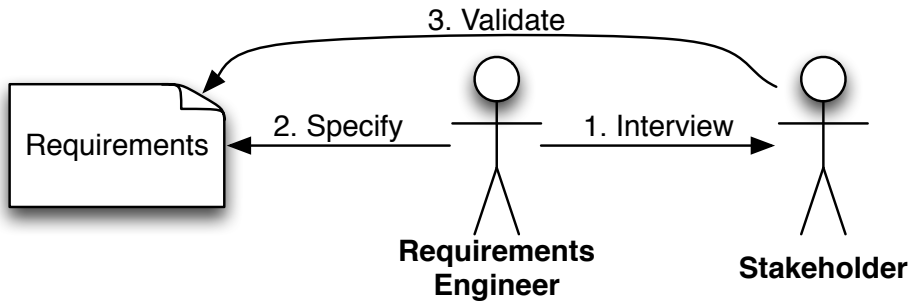


Figure 1.1.: A requirements engineer iteratively interviews different stakeholders (1), specifies their individual requirements (2), and later on the stakeholder can validate whether the requirements engineer's specification is correct (3)

interpretation of what a stakeholder said can be wrong because of a miscommunication between them or due to an incorrect assumption of the stakeholder. Either way, any of the requirements that the requirements engineers collect can be incorrect and, consequently, inconsistent with the other requirements. Of course, inconsistencies may also arise if different stakeholders with diverse perspectives are interviewed and their statements are in conflict. Only through requirement validations not only with the originally interviewed stakeholder but also with other ones (cf. Figure 1.1), the requirements engineer can ensure that incorrect requirements are corrected and inconsistencies are resolved. As pointed out by Zowghi and Gervasi [ZG03], increasing the consistency of a specification may require the removal of distinct requirements which, in turn, decreases the overall completeness of the specification. They further point out that the introduction of additional requirements increases the completeness but, at the same time, may decrease the overall consistency of the specification.

The requirements engineers collect the requirements iteratively alongside the development, or they collect them in a Software Requirements Specifications (SRS, cf. [IEE98]) which is handed over to be realized in a follow-up phase [Pre05, Som06]. As argued by Alexander and Beck, this choice depends on the size and scope of the system that is being specified and developed [AB07].

1.1. Establishing Correct, Consistent, and Complete Requirements

Creating a collection of correct requirements which are consistent and completely cover the intended software system [Poh93] would be quite easy if all stakeholders would always provide objective and correct answers to what they require and if they would do so without contradicting other statements. In practice, however, this is not the case due to stakeholders being subjective in their statements and the fact that for complex

software systems there is not one distinct stakeholder who can describe everything that the software engineers require to know in order to build the right system.

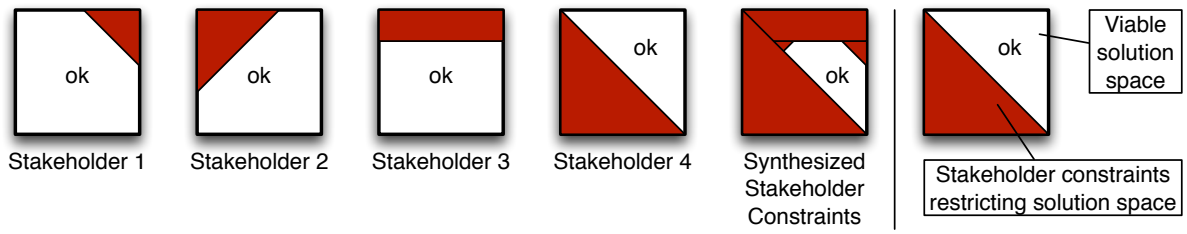


Figure 1.2.: Requirements from different stakeholder groups (views one to four, respectively) need to be synthesized (fifth view) in order to know which solutions are viable (adapted from [Dav93])

Since each stakeholder has an individual, subjective perspective on what is required, synthesizing their perspectives to produce a specification that all stakeholders can agree upon is a challenging task even for experienced requirements engineers. As illustrated in Figure 1.2, each group of stakeholders has specific needs and requirements which restrict the solution space for the final product [Dav93]. Consequently, all stakeholders need to be interviewed since the knowledge about what needs to be supported is fragmented and "distributed in several involved people's minds" [Des08]. Of course, each stakeholder may also have arguments for or against statements made by other stakeholders [JMF09]. This becomes even more challenging for the requirements of innovative products, since these are inherently elusive and hard to describe [Poh10]. Only by eliciting all requirements and checking them against each other, can the requirements engineers synthesize a specification of a software system that supports everybody who is impacted.

As in the story of the blind men and the elephant [GC04], each man perceives and describes a different part of the animal. Since each stakeholder belonging to another group of stakeholders [Ale05] would describe different aspects, forgetting to talk to a stakeholder leads to an incomplete specification which does not cover all the required conditions and capabilities. Additionally, as argued by van der Aalst [Aal07], people tend to oversimplify how stakeholders work together. Having abstracted too much from reality, the requirements engineers might end up with what he refers to as *PowerPoint reality*. Consequently, it is up to the requirements engineers to interview all stakeholders and to derive a specification which describes a software system, which conforms to all of the individual stakeholder perspectives. Only then can it be ensured that the software system is accepted and later used by all stakeholders.

1.2. Problem Definition

As previously stated, requirements engineers have to talk to many different stakeholders. After they captured statements of individual stakeholders, they then have to validate

the stakeholder requirements generated from these statements. As well as validating these requirements with the stakeholder they originated from, the requirements engineers must seek feedback from other stakeholders who might be affected. Of course, the more stakeholders are involved, the more effort is necessary to talk to all of them to validate requirements or elicit new ones. These interactions can take place either *individually* or in *groups* of up to all stakeholders involved in a particular scenario. This decision depends on factors such as the stakeholders' availability, which usually differs, and their distribution over different locations. Consequently, we arrive at the first problem to be tackled:

Problem P_1: Interacting with all involved stakeholders – managing availabilities, scheduling dates, traveling – is difficult, especially since most of their requirements are interconnected and potentially in conflict.

Both types of stakeholder interactions, engaging one stakeholder at a time or groups of them simultaneously, have distinct advantages, most of which solve problems of the alternative setting, e.g. fragmented scenarios in individual stakeholder sessions or the influence of hierarchies in a session engaging multiple stakeholders. By combining both approaches, i.e. by providing the advantages of group sessions in isolated individual sessions, we hope to provide a better solution for the trade-off involved when deciding which type of session should be scheduled next.

Thesis-Goal TG_1: The first goal of this thesis is to enhance individual sessions with one stakeholder in such a way that *A)* the stakeholder gets immediate feedback on his or her statements based on (previously captured) statements, responses, or behavior of other stakeholders and that *B)* the stakeholder is enabled to directly build on and complement the scenarios of other stakeholders ($\mathbf{TG}_{1A}$ and $\mathbf{TG}_{1B}$, respectively).

For a stakeholder to be able to complement an incomplete scenario as described by other stakeholders, he has to understand which parts of it have already been captured. Therefore, an approach trying to enhance individual sessions must overcome the fragmentation of scenarios that is inherent in this type of stakeholder interaction. Only then would an isolated stakeholder be able to validate whether what was captured is correct from his point of view and how it might be complemented.

Powell [Pow10] compared the elicitation of requirements to collecting individual pieces of a giant jigsaw puzzle from different stakeholders. While a scenario which involves multiple stakeholders and has many alternatives can be considered equivalent to such a jigsaw puzzle, the activities of individual stakeholders within this scenario represent the corresponding pieces – each one has a distinct precondition which needs to be fulfilled by the postconditions of other activities. During the collection of these pieces, the requirements engineers rarely know what the complete puzzle looks like and how many

pieces there are. Each individual session provides new pieces which are difficult to match against any of the others without first knowing the context of either of these pieces. Consequently, by providing an understandable visualization of which pieces of the puzzle are already captured and how they might fit together, even an isolated stakeholder would be able to pinpoint errors (e.g., which ones do not fit or are incorrect), and fill the gaps between some of the already captured pieces by providing additional pieces or rearranging existing ones. In short, the statements and requirements of stakeholders in individual sessions would not be isolated pieces of a big puzzle anymore.

It would hardly be feasible for requirements engineers to create such a visualization manually for each stakeholder or each session. It is possible, however, to automate this creation if the pieces are unambiguous and well-defined. Generally, formal models help engineers to cope with the complexity inherent in engineering endeavors – in this case, to define fragmented pieces of a scenario in which many different stakeholders are involved. Still, employing formal models to capture requirements from a stakeholder who later has to validate whether these are correct introduces a new problem:

Problem P_2: Formally defined requirements cannot directly be specified, validated, or corrected by stakeholders. Instead, such models have to be explained or translated for stakeholders, and may only be modified directly by requirements engineers, who also have to translate stakeholder feedback into model updates.

To identify incorrect requirements, stakeholders have to judge the correctness of the gathered requirements. Depending on the formality of the requirements' presentation, translation overhead is introduced to ensure that stakeholders understand the requirements.[2] As Luebbe and Weske [LW11] point out, to elicit the scenarios and interactions between stakeholders, a method expert who is familiar with a formal modeling notation elicits the collaborative scenarios and later creates a corresponding model. In a subsequent stakeholder session, this model is then presented, discussed, and iterated – usually relying on face-to-face explanations. Alternatives for assisting stakeholders in understanding formal models include natural language annotations, redundant informal representations [ARE96], or the creation of prototypes that the stakeholders can experience directly [SS97].

Tangible prototypes are a characteristic part of innovation-oriented processes such as *Design Thinking* [CS08, Bro09, PMW09], and they support the collection of requirements or the iteration of ideas with numerous stakeholders in a feasible manner. Designers employ prototyping not only to explore possible solutions or design alternatives – rather, prototypes are used as early as possible to explore and define the stakeholders' needs which any suitable solution should address. Inspired by tangible prototypes, which can intuitively be understood and changed by prospective end users, we propose model-based

[2] This overhead is among the reasons why Davis [Dav93] argues, that requirements should only be specified formally if "we cannot afford to have the requirement misunderstood" by software engineers.

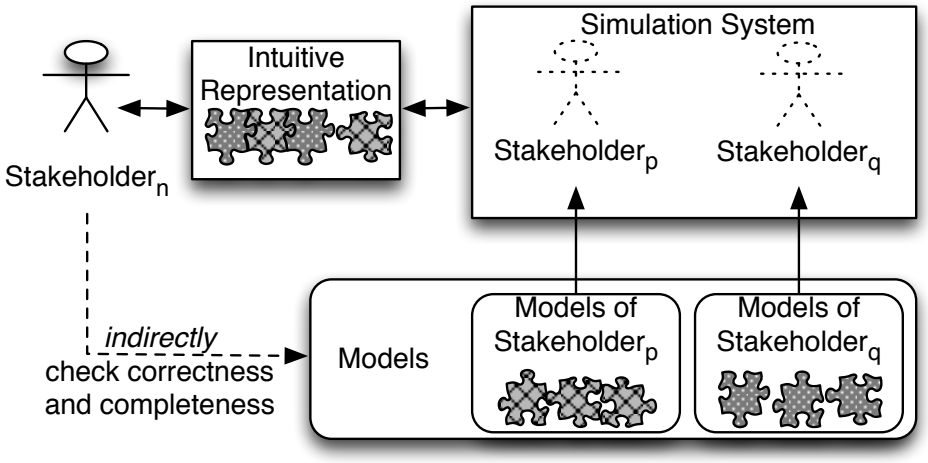


Figure 1.3.: Individual stakeholders are able to validate pieces of relevant scenarios if these pieces are represented in a fashion that is simple to understand, as envisioned in $\mathbf{TG}_2$

virtual prototypes. Through being able to automatically derive such a prototype of what the requirements engineers have captured so far, stakeholders can directly experience how they were understood without the requirements engineers having to invest time and effort into the creation of the prototype.

Thesis-Goal $\mathbf{TG}_2$: Based on formally defined behavioral models, our goal is to create a prototyping approach *A)* which allows stakeholders to understand such models intuitively, *B)* which allows them to initially build and modify these models intuitively, and *C)* which allows requirements engineers to quickly and inexpensively generate and iterate behavioral models. These three points are discussed below:

$\mathbf{TG}_{2A}$) In order to be able to represent the content of the captured models intuitively for stakeholders, whose collaborative scenarios have to be elicited, we have to rely on the highest common factor between these stakeholders. Since this factor can only depend on how the stakeholders interact in the workplace, such a representation has to reflect this workplace as well as its contextual triggers [SMK+09]. Our approach is aimed at eliciting and validating behavioral models which capture pre- and postconditions of stakeholder activities and their interactions among each other.

$\mathbf{TG}_{2B}$) Stakeholders have to be able to modify a virtual prototype similar to how they would change a tangible one. Hence, stakeholders interacting with a prototype derived from a set of behavioral models should be able to complement these models by intuitively adding new ones or identifying erroneous ones themselves to correct them.

$\mathbf{TG}_{2C}$) To ensure that the use of formal models becomes feasible, the automation affordances of formal models have to be exploited. As pointed out by Pohl [Poh10], prototyping can be done quite inexpensively and quickly if the prototypes can be *gener-*

ated.[3] Unfortunately, the more interactive, the more complex and the more expensive a prototype becomes [LW99]. By automatically generating prototypes which are suitable for stakeholders, our approach can ensure quick and inexpensive iterations.

1.3. Contributions

The conceptual contributions of this thesis are as follows. The main contribution of this dissertation is the concept of virtual model-based prototyping for behavioral specifications capturing stakeholder actions and interactions as they occur in collaborative stakeholder scenarios. Inspired by tangible prototypes and the similarity between models and prototypes, we developed a state-based simulation approach with an animated visualization that is intuitive for stakeholders in such a way that they understand the behavioral models underlying the simulation. Thereby, the stakeholders are enabled to validate and provide feedback about models which originated from other stakeholders. This simulation relies on graph transformation systems which are more concrete than automata or sequence charts used in related approaches. Furthermore, by deriving a virtual prototype based on the already captured models, it becomes possible to establish an overview of which pieces of the overall puzzle have already been captured, even for individual stakeholders participating in isolated sessions. Based on such an overview of how the collected behavioral models may fit together, immediate feedback based on prior sessions can be provided, similar to a group session. Additionally, the context for activities or interactions, observed from such a session, is already established as well.

As our second contribution, we present our simulation algorithm which guides stakeholders through captured scenarios in which they are involved, so that they can validate specific situations and their respective actions as well as explore scenarios that were not yet covered. To overcome stalemates due to seemingly non-deterministic decisions of other stakeholders during an elicitation with an affected stakeholder, our approach supports the definition of what we refer to as *expected continuations*, which allows the impacted stakeholder to define fragments of how a scenario in such a stalemate should continue. A black box abstraction for stakeholders concerning their reactions on specific inputs is introduced which, in turn, enables impacted stakeholders to walk through their scenarios even though interactions with other stakeholders have not been observed and captured yet. By matching such expectations to the pieces of the puzzle that other stakeholders contribute to the scenario puzzle, the requirements engineer eventually ends up with the synthesized collaborative scenario or identifies a potential conflict. Either way, this matching approach overcomes the fragmentation resulting from individual stakeholder sessions by automatically exploring how they might fit together and validating the resulting combinations with the stakeholders.

[3] Pohl even rates prototypes which can be generated as requiring the least effort of all validation techniques he discusses in [Poh10].

Additionally, we present an evaluation of our concepts based on a research prototype which includes a domain-specific graphical user interface that maps and illustrates the side-effects of story patterns in an intuitive manner for stakeholders. To decouple stakeholders locally and thereby reduce the requirements engineers' need for traveling, the implementation is web-based and enables remote scenario simulations. The evaluation covers the understandability of the implemented virtual prototyping approach, quick and inexpensive iterations, and the stakeholders' ability to understand and change the models underlying the prototype.

The contributions of this thesis illustrate the feasibility of virtual model-based prototypes to support requirements engineers in eliciting and validating collaborative scenarios as they take place between numerous stakeholders. Through these contributions, some of the most important advantages of workshop-like group settings involving multiple stakeholders have been transferred to individual sessions taking place in isolation.

The contributions were inspired through the insight that *tangibility* of prototypes simplifies communication between stakeholders and requirements engineers – an insight which we owe to our investigations into the innovation process *Design Thinking* and which was part of the overall frame in which this thesis and its contributions were developed.

1.4. Structure

Chapter 2: The preliminaries of this thesis are discussed in this chapter. Specifically, this includes the activities related to requirements engineering, i.e. how requirements can be elicited, modeled and validated. Also, the suitability of prototypes to validate assumptions and requirements is discussed. Finally, graph transformations are introduced as a structural modeling technique suitable to describe behavior.

Chapter 3: We present an overview of our approach to ease the understanding of the forthcoming chapters. Furthermore, this chapter discusses the similarities of models and prototypes and how we can bridge the remaining gap using *virtual prototypes*.

Chapter 4: This chapter discusses how our simulation approach replays behavioral models (i.e. *story patterns*), illustrated using the case study of selling a movie ticket.

Chapter 5: In this chapter, we discuss how we extended our approach to include the elicitation of story patterns based on observed stakeholder interactions with a virtual prototype.

Chapter 6: This chapter introduces *triggers* and *expected continuations* which support stakeholders to overcome *stalemates* during the elicitation.

Chapter 7: The concepts presented in the former chapters are partially implemented as a research prototype which is discussed in this chapter.

Chapter 8: This chapter presents an evaluation of the understandability of the realized *virtual prototype* and the stakeholders' ability to modify it. Additionally, the costs and the effort required to build, i.e. derive and start, a session are discussed.

Chapter 9: The state of the art of simulating and visualizing requirements for stakeholders and how these approaches are integrated into the validation is discussed in this related work chapter.

Chapter 10: Finally, we draw conclusions and discuss directions for future work.

2. Preliminaries

Alongside the requirements engineering activities illustrated in Figure 2.1, this chapter discusses the preliminaries that this thesis builds upon. Starting with the discovery of stakeholders and new requirements, the resulting insights are modeled and documented. Afterwards, the documented requirements engineers' interpretation of what the stakeholders said is validated with the stakeholders again to ensure that it correctly reflects what the stakeholders intended to say, i.e. what they need. As illustrated in Figure 2.2, the requirements engineers arrive at a *complete* specification if it contains and specifies all stakeholder needs ($A \setminus B = \emptyset$) [Dav94]. To arrive at a complete and *correct* specification, the requirements engineers also need to ensure that the documentation does not contain incorrect requirements which do not reflect stakeholder needs ($C \setminus B = \emptyset$).

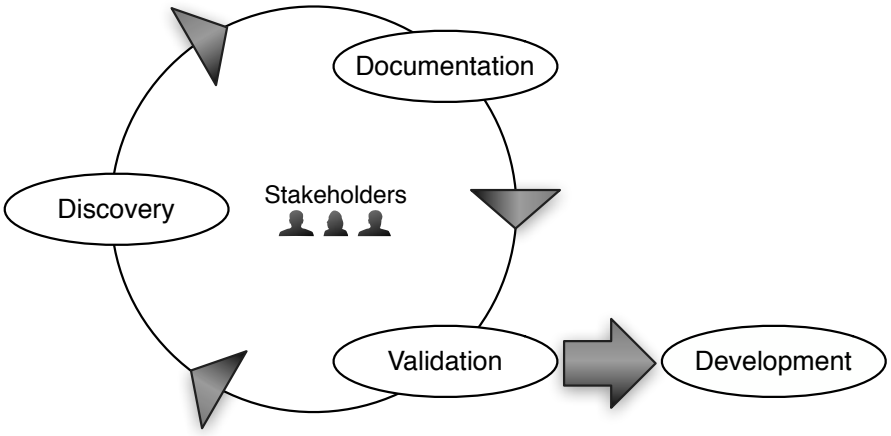


Figure 2.1.: Inquiry circle of requirements engineering activities (adapted from [ABD09])

Requirements engineers have to produce a requirements specification document that is consistent, complete and contains only correct requirements.[1] The validation (whether individual requirements are correct) as well as the elicitation and exploration of all scenarios to ensure that the gathered set of requirements is complete depend on the interaction with stakeholders. The consistency of a set of requirements, on the other hand, is usually achieved without stakeholders, since consistency issues relate to the conflicting

[1] Although more detailed lists of quality attributes which requirements have to possess exist [Dav05, BPKR09], these three are considered the most relevant for a requirements specification.

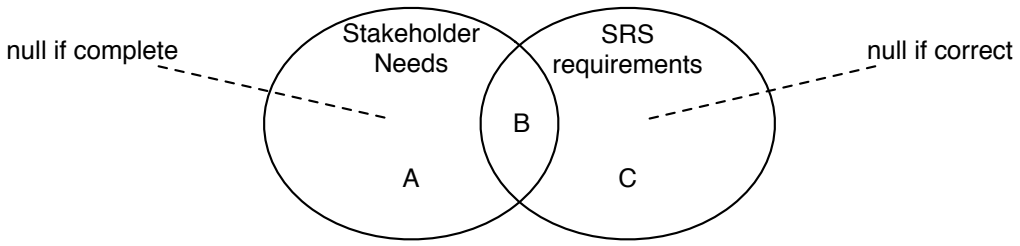


Figure 2.2.: Requirements engineers interact with stakeholders until B=A∪C (adapted from [Dav94])

statements of different stakeholders. Inconsistencies can be identified by checking each requirement against all others to ensure that they do not contain conflicting statements. While this procedure can be executed in more structured ways to compare only related requirements, a complexity of $\mathcal{O}(|Requirements|)$ for checking each new requirement remains.

Choosing an informal specification, such as (structured) natural language, only very limited tool support is available to automate consistency checks, e.g. [GZ05]. Consequently, such checks are manual, time-consuming and error-prone [AS02]. This is one of the main advantages of employing formal modeling notations to specify requirements. While formal models assist requirements engineers in checking the consistency of their requirements, such models are disadvantageous for tasks relying on stakeholder interactions. Specifically, they restrict what can be expressed due to their metamodel (cf. [GGLS11]) and formal notations have to be explained to stakeholders [ARE96] to enable them to validate whether their intentions were correctly conveyed in these models.

The requirements engineering process is discussed in detail. First, Section 2.1 discusses how requirements are elicited from stakeholders. Then, the specification of requirements is described in Section 2.2. Since stakeholders need to validate the requirements, Section 2.3 lists the requirements engineers' options of including stakeholders in the validation of their requirements. Afterwards, Section 2.4 illustrates the usage of *scenarios* in requirements engineering. Section 2.5 concludes this chapter with a description of *graphs*, *graph transformations*, and *story patterns*, which are the formalism chosen for this thesis.

2.1. Elicitation and Discovery

As argued by Endres and Rombach [ER03], it is "good practice to start a requirements process with a vision statement." Then, the requirements engineers can interview all stakeholders and establish this vision in a broader context [JP93]. This entails the elicitation of individual *statements of needs*, which have to be specified and converted into *stakeholder requirements* [HJD11]. Iteratively, the requirements engineers gather knowledge about the stakeholders' domain and its properties (*dom*) as well as requirements (*req*) based on these properties [Jac00]. Only then are software engineers later able to

design a specification (*spec*) in such a way that it fulfills these collected requirements: $spec, dom \models req$.

Apart from stakeholders, other sources for requirements are also available [AS02, PR11], although most of them are static and out of scope for this thesis – these sources cannot provide feedback on whether the requirements engineer's interpretation is correct or not. Consequently, the most important source of information is stakeholders, especially the ones who are prospective users of the software system that is specified. To interact with stakeholders, Davis et al. found in a literature review that structured interviews are a method which, although not the best in all situations, is always suitable and always yields good results [DDH+06]. Still, to gather requirements, the interactions between requirements engineers and stakeholders take place either in *groups* of up to all stakeholders involved in a particular scenario or *individually*.

2.1.1. Group Sessions

Different types of group sessions exist, not only for eliciting [LW99, AS02, Lau02, MNK+07, Lue11], but also for validating requirements [SS97, RS09, Poh10]. In such group settings, hierarchies among the participants can influence the elicitation [Mur11] in terms of whether specific conflicts are addressed, how decisions are made, and how conflicts are resolved. Apart from such group dynamics, group sessions also introduce scheduling overheads. The more people attend, the more complex it becomes to find a time slot and a location all participants actually agree on. In case of globally distributed stakeholders, it may even be infeasible to get all stakeholders physically together for such a meeting. Therefore, technical settings such as video conferencing exist, which decouples the participants from a specific location [DESG00, Dam01, LRA02]. Still, even with techniques for improving synchronous communication in distributed settings [CDL07], time zone issues remain and stakeholders "continue to rely on *formal* channels" or asynchronous channels for communication [Dam07].

However, group sessions are quite effective for quickly eliciting, validating, and iterating scenarios [LW11], since each stakeholder statement may instantly be countered by alternative arguments until a consensus is found. Thus, all participants in such a meeting can arrive at a commonly shared understanding of their collaborative scenarios and requirements.

To sum up, while instantaneous feedback provided by each other allows stakeholders to iterate their understanding, group sessions are infeasible to conduct often, if at all.

2.1.2. Individual Sessions

In an individual elicitation session, an isolated stakeholder can describe only the externally visible behavior of any other stakeholders he interacts with. The actual behavior

of these interaction partners has to be considered as a *black box*.[2] Consequently, without yet knowing what the interaction partner bases her decisions on or what her private policies [DMCS05] are, the interviewee is likely to describe non-deterministic behavior, e.g. different responses for seemingly identical requests, from any of his interaction partners. We refer to such points during the elicitation of a scenario as a *stalemate*: without knowledge of how an interaction partner reacts, the stakeholder can only speculate on what happens next. Therefore, the investigated scenario remains incomplete until the opposite perspective is elicited to complete the interaction.

As opposed to group sessions, the requirements elicited in individual sessions are rather *fragmented*. After each session, the overall scenarios which were already captured have to be amended with the new fragments. Thus, the incidental complexity [NS00] of the requirements engineers' models increases more rapidly, since smaller chunks of information are added or updated more often than in group sessions. Still, individual sessions are well-suited for tasks such as resolving an identified conflict, exploring alternatives of a specific scenario, or validating another stakeholder's results. Also, short individual elicitations may be the only way of getting to talk to stakeholders whose availability is restricted due to time or budget constraints.

Correcting an error in the requirements or resolving a conflict between requirements has to be possible during any short, iterative sessions with the stakeholders that were misunderstood. Still, visiting these stakeholders individually might require the requirements engineers to travel quite often, as discussed in our report of how employees of D-LABS GmbH have to travel all over the world to meet their stakeholders [GGS11a]. These traveling times may be reduced by decoupling stakeholders and requirements engineers from one location as discussed for the group sessions.

Maiden et al. [MNK+07] report gathering seven times more requirements per person per hour in individual workplace sessions than in group sessions. Additionally, sessions with individual stakeholders inherently emphasize the individual viewpoint of the interviewee [SS97], which in turn allows stakeholders to discuss their needs freely without being restricted by hierarchical constraints.

To summarize, talking to individual stakeholders to elicit their collaborative scenarios may be the only possible method of talking to *all* stakeholders, but it may lead to fragmentation of these scenarios. Since each session may require additional sessions to validate new insights, it is usually not foreseeable how many sessions will be necessary.

2.2. Documentation, Specification, and Models

Based on stakeholder statements which the stakeholders usually expressed in *natural language* (NL), the requirements engineers have to extract and specify requirements. While

[2] "A usually complicated electronic device *whose internal mechanism is usually hidden from or mysterious to the user*; broadly: *anything that has mysterious or unknown internal functions or mechanisms*" [Mer13]

NL can be assumed to be common to everybody, it is inherently ambiguous to deal with [GZ05]. Although there exist many shortcomings of NL requirements, different studies indicate that most requirements engineers still rely on NL for expressing requirements [NL03, LMP04]. As Dawson and Swatman [DS99] argue, *informal models* are considered to be models that "can be understood and explained without specific training". Further examples are "natural language models including text descriptions, use case scripts, ad hoc diagrams and interactive demonstration models as often produced for prototypes".

Formal models, on the other hand, can be used to unambiguously express what stakeholders require. Since they have to conform to a standardized metamodel, they are limited in terms of what they are able to express (syntactical restrictions). Still, this also ensures that everybody who is familiar with the model's corresponding modeling notation and its concepts can unambiguously understand what the modeler intended to express. However, employing a formal modeling approach introduces a barrier between the stakeholders and their requirements – since formal models are considered to "require training in order to be understood or explained" [DS99]. Therefore, without either additional annotations or even face-to-face explanations, stakeholders cannot understand, judge and comment on these requirements [ARE96]. Additionally, there is not *one* type of model which can be used to capture *everything* that stakeholders require as argued by Alexander [Ale11]. Of all the real world aspects which have to be considered, each one of these notations abstracts most of these aspects while emphasizing a specific subset. Consequently, each notation overcomes specific disadvantages and is useful for a specific purpose (cf. Pohl's definition of *model* [Poh10]). Still, to be able to derive prototypes automatically and feasibly, adherence to a formal metamodel with a guaranteed semantic is necessary. Thus, informal requirements are not within the scope of this thesis.

The Unified Modeling Language (UML) provides multiple models for different purposes [BHK04], mainly for modeling *structure* (e.g. class and object diagrams) and *behavior* (e.g. sequence diagrams, use case diagrams, and state machines). For tasks related to requirements engineering, use cases [KG00, Coc01] and class diagrams are used quite frequently [SL04, DP06]. Since class diagrams can be used quite easily to model the concepts which can be found in the stakeholders' domain and how these concepts are interconnected there exist different approaches for either generating class diagrams from natural language statements [AG01] or vice versa [MAA08].

UML and Metamodeling: A UML class diagram prescribes and defines all valid states which can be modeled and instantiated by a UML object diagram [OMG05a, OMG05b]. Kühne [Küh06] argued that a metamodel of a domain establishes a language to be used to construct sentences describing situations as they can be observed in this domain. Consequently, after a specific model and its notation are chosen, *all* possible situations

can be expressed using these concepts, but only concepts defined by the corresponding metamodel can be employed using this notation.[3]

Cranefield et al. propose [CP99, KCH+02] and demonstrate [CHP01] the usage of UML class diagrams to model ontologies. An *ontology*, in our case, is an explicit specification of a conceptualization [Gru93]. Furthermore, Uschold [Usc98] defines an ontology as necessarily consisting of "a vocabulary of terms and some specification of their meaning" which also includes "definitions and an indication of how concepts are interrelated". Thereby, they "collectively impose a structure on the domain and constrain the possible interpretation of terms." Throughout this thesis, metamodels are primarily considered as *world view* instead of a mere *abstract syntax* for models [LK10]. Thus, the terms *ontology* and *metamodel* are used synonymously throughout the remainder of this thesis.

2.3. Validation of Models

Each model that is created based on stakeholder statements is rather a requirements engineer's interpretation of what the stakeholder may have intended to say [Poh10]. In general, individual requirements can have different errors as illustrated in Figure 2.3. These errors, such as ambiguity or incorrectness, may be due to misconceptions of the stakeholder, misunderstandings between requirements engineer and stakeholder, or a misinterpretation during the creation of a requirement. Since these error sources cannot explicitly be excluded with certainty, the requirements engineers have to ensure that each requirement gathered is correct, feasible, unambiguous, verifiable, modifiable, and traceable [BPKR09].

Furthermore, requirements are not set in stone. Instead, they only provide a snapshot of what was required at the time they were elicited [Dav05]. They are subject to change due to a number of reasons such as new technologies, new regulations, or new products released by competitors [PR11]. Additionally, requirements and their importance for stakeholders *age* – priorities change and *satisfiers* which were explicitly demanded are later on just subconsciously taken for granted (cf. *Kano model* as described in [PR11]). Consequently, eliciting requirements over a long period of time implies that these requirements have to be validated more often.

Finally, all collected requirements have to be consistent and complete [BPKR09]. Sets of requirements may contain additional errors as pointed out by Avci [Avc08] (cf. Figure 2.3). A requirement introduced into a set of existing ones may be redundant or inconsistent, i.e. in conflict with another one. Both of these problems can be addressed automatically with available tool support (e.g. [GZ05, SGM05]).

To ensure that the requirements are suitable, i.e. possess the quality attributes mentioned above, the requirements have to be "validated (checked informally) as acceptable

[3] As part of the *FlexiTools* workshop series, on the other hand, Ossher et al. argue that choosing such a notation at an early point in time is a strong and restricting commitment [OHS+10] and, therefore, that modeling tools should explicitly support *ad hoc* modeling [AAF+09].

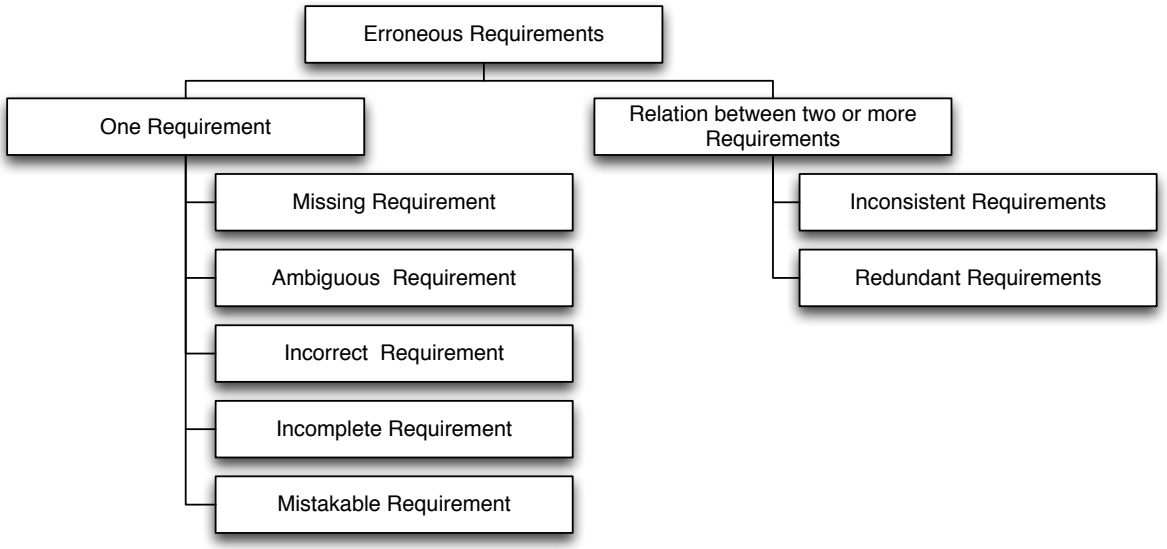


Figure 2.3.: Types of requirements errors (adapted from [Avc08])

to the customer" [ZJ97]. The requirements engineers also have to ensure that the set of collected requirements "as a whole has been validated as expressing all the customer's desires with respect to the software development project". Again, individual or group sessions can be conducted to validate requirements. Group sessions, as Pohl points out [Poh10], can be quite formal and complex to set up. In such a session, a requirements engineer may provide a face-to-face explanation [ARE96] of the formal requirements he defined to enable stakeholders to understand these requirements.

In 2009, Cheng and Atlee pointed out the current research directions in requirements engineering [CA09]. They highlighted *simulation* and *animation* as methods which would help not only during the elicitation, but also during the validation of requirements. While both of these techniques are necessary for creating interactive (simulation) and understandable (animation) representations of requirements which stakeholders can validate, the remainder of this section focuses on visualization and animation approaches, as well as prototypes in detail.

2.3.1. Visualization and Animation

To be included into the validation of requirements based on their statements, stakeholders have to understand these requirements. If the requirements were specified formally, it is necessary to provide a representation which can be easily understood. For requirements engineers, visualizations aim at "mapping complex data to perceptual representations in such a way as to maximize human understanding and communication" [GMM07]. Stakeholders, on the other hand, require a representation that overcomes the abstraction introduced by the modeling notation and its concepts. Consequently, the

content of such models needs to be mapped "into visual artifacts, permitting stakeholders to actually *see* the requirements", e.g. using metaphors stakeholders are familiar with [GMM07].

While visualization mainly conveys the structure and relationships [GMM07], animation illustrates the dynamic behavior instead. The animation of a specification, as defined by Lalioti and Loucopoulos [LL93], is "the process of providing an indication of the dynamic behavior of the system" by walking through parts of the specifications "to follow some scenario". Thereby, animation can be used to "determine causal relationships", that are embedded which stakeholders would not perceive otherwise, and to "ensure adequacy and accuracy by reflection of the specified behavior back to the user".

As pointed out by Gemino [Gem04], visualization, animation, and even narration enable stakeholders to better understand requirements. Still, when choosing how to visualize requirements, it has to be noted that *a)* the chosen representation [SS99] and *b)* the stakeholder who sees it [SML+09] influences which aspects are primarily perceived and, thus, what kind of feedback can be obtained.

2.3.2. Prototyping

Based on elicited assumptions and requirements, the requirements engineer may prototype these to ease the stakeholder validation. Prototypes can be distinguished as *tangible* and *intangible* ones. While the former kind is usually rather low-fidelity and easy to build, the latter, although more expensive to create, is the only viable option for abstract concepts and "things you cannot touch" [Bro09] such as how an innovative service, which does not yet exist, might respond to an end user's input. In the following subsection, which is partially based on [GEGS10] and [GGS11a], both kinds of prototypes are discussed.

Tangible Prototypes in Design Thinking: In innovation processes such as *Design Thinking* [CS08], assumptions and ideas are usually externalized and communicated using tangible prototypes in physical manifestations [LST08]. Prototyping is an important method of validating assumptions about stakeholders and provoking reactions from them. In engineering, prototyping typically aims at creating tangible representations of design ideas. Additionally, prototypes can help to establish a common understanding between design thinkers and end users as well as between design thinkers themselves. Furthermore, prototypes can facilitate discussion during divergent project stages and agreement in convergent stages of a project. Depending on how well stakeholders can experience and perceive a prototype, they are able to judge and evaluate the design idea or even the rationale behind it.

For Brown [Bro09], a prototype is "anything tangible that lets us explore an idea, evaluate it and push it forward". Most importantly, physical prototypes usually afford stakeholder modifications – by being able to change the prototype themselves, they can mold it to fit their specific perception of how it should behave or look like.

Prototypes in Software Engineering: In software engineering, tangibility for stakeholders is quite difficult to achieve for designs or ideas concerning inherently complex intangible or non-material concepts. Current prototyping practices of software systems development focus on the articulation of graphical user interfaces (GUI) using a broad range of approaches from paper prototyping [Sny03] to evolutionary prototyping implementations that evolved based on evaluations till the prototype is suitable for production [BBLZ96]. However, these prototypes usually represent the perspective of an individual and isolated stakeholder. Therefore, such prototypes are not suitable to elicit feedback about the underlying concepts of how activities are executed and intertwined or how an innovative software solution could support them. Instead, these prototypes mainly address usability issues. Since software engineers can judge whether a system is built correctly and without errors, they can hardly answer whether the right system is built, i.e. suitable for the task it was designed for, since only a holistic view of all the intertwined perspectives could answer this question.

Thus, while the stakeholders can judge whether the system is built correctly, i.e. usable for them, they can hardly answer whether the right system is built, i.e. suitable for the task it was designed for, since only a holistic view of all the intertwined perspectives could answer this question.

Prototypes provoke discussions, between stakeholders and developers as well as among developers. Sommerville and Sawyer [SS97], and Pohl [Poh10] point out that prototypes are suitable for increasing the stakeholder involvement in the elicitation and validation of their requirements – but only as long as their creation is inexpensive and allows for quick iterations. However, software prototypes may become "the private toys of their developers" which means that "no serious attempt is made to capture all the insights sparked by prototypes", as Schneider argues [Sch96]. Thus, it is just as important to capture the discussions about the prototype as it is explained by its developer, since only the "*developer-prototype system* represents the full knowledge".

2.4. Scenarios

Glinz [Gli00] defines a scenario as "an ordered set of interactions between partners." Further, an *instance scenario* may "comprise a concrete sequence of interaction steps" (e.g. a specific story of one stakeholder), while a *type scenario* consists of "a set of possible interaction steps" (e.g. the result of a synthesis of multiple instance scenarios). As pointed out in literature, (*instance*) scenarios support stakeholders to structure their thoughts along a narrative [ABD09]. Further, scenarios allow requirements engineers to provide a context for specific requirements which would be hard to evaluate without a corresponding context – especially if innovation is involved, scenarios are a suitable way of making the *innovation* tangible [Bro09, OP10].

As Cheng and Atlee point out [CA09], although scenarios are inherently incomplete, they are quite easy to use for requirements engineers and stakeholders throughout the

whole requirements engineering process. During the elicitation, domain knowledge needs to be gathered. Thereby, scenarios provide a vocabulary to communicate with stakeholders [Car95], since scenarios are always from the stakeholders' perspective [Kuu95]. Further, since each scenario is related to the modeling of a "very restricted part of the system's behavior at a time", requirement engineers can focus without "getting lost in the complexity of tackling the entire behavior" [HD98]. Consequently, scenarios are inherently incomplete as perspectives of other stakeholders are usually missing – hence, the fragmentation in individual sessions (cf. Section 2.1.2). Therefore, to complete a partial scenario the requirements engineer has to synthesize multiple partial scenarios from different stakeholders similarly to how pieces of a puzzle need to be arranged.

Modeling approaches for scenarios range from using structured natural language in *use case templates* [Coc01] to formally modeled flows of activities as in *story-driven modeling* [DGZ04]. Finally, scenarios can be validated directly by stakeholders, since scenarios are structured similar to stories. Hence, they afford the requirements engineer to ask questions to ensure correctness (*Are these steps in the right order?*, *Can this actor carry out this action?*) and completeness (*Is this story complete?*, *Has a step been omitted?*) of requirements [AM04].

Based on the support scenarios offer throughout the requirements engineering activities, their inherent incompleteness, and the ability to cope with complex collaborations between different stakeholders, the approach presented in this thesis aims at collecting collaborative scenarios.

Play Engine: One of the most prominent scenario-based approaches for the specification of reactive systems is Harel and Marelly's *Play Engine* [HM03a]. Their approach uses *Live Sequence Charts* (LSC [Har01]) which distinguish between possible and necessary behavior. By providing a prototypical mock-up of a software system's user interface, a domain expert can *play in* desired behavior, i.e. scenarios of how the system's interface has to react to user inputs [HM03b]. For instance, pressing a specific button must change the color of a distinct label. In the background, the play engine automatically generates "formal requirements in the language of LSCs" which capture the changes of the user interface that the domain expert demonstrated – *observation-based*, "without a need to explicitly prepare the LSCs" [HKMP02].

Play-out, on the other hand, is "the process of testing the behavior of the system by providing any user actions, in any order, and checking the system's ongoing responses" [HM03a]. During play-out, a stakeholder provides an input and the Play Engine tries to find a suitable reaction as specified in the LSCs that constitutes an acceptable continuation for the stakeholder. Since the scenario-based specifications are inherently incomplete and, hence, rather fragmented, multiple specifications may cover different but *overlapping* fragments of the same scenario. Therefore, during play-out sessions, the Play Engine may nondeterministically choose one of multiple alternatives which the LSCs define. To reduce this nondeterminism, Harel et al. [HKMP02] employ model

checking to check whether a *correct* next step exists among the possible continuations which the LSCs contain. They refer to the resulting, "strengthened execution mechanism" as *smart play out* [HM03a]. Consequently, even if conflicting LSCs exist which result in mostly invalid states during the play out, *smart* play out ensures that the follow-up state is a valid one (if a valid continuation exists).

Throughout this thesis, to *play out* a behavioral specification will be used synonymously with replaying an executable behavioral specification (which was derived from a prior observation). Further, to *play in* a specification refers to a stakeholder who executes an activity to progress a scenario, i.e. changes the state of this scenario, in such a way that a specification representing these changes can automatically be derived by comparing the states of the scenario before and after the stakeholder acted.

2.5. Graph Transformations

To specify graph-based model transformations, Grunske et al. [GGZ+05] define *Directed Typed Graphs* as follows: T_V and T_E are sets of vertex types and edge types, respectively, and $\mathcal{G}$ is the set of all possible graphs over T_V and T_E. From this set $\mathcal{G}$, a directed typed graph $G = (V, E, src, trg, type)$ has two finite sets V and E of vertices and edges. Further, the two functions $src : E \to V$ and $trg : E \to V$ assign a source and a target vertex to each edge, respectively. The function $type : V \to T_V \cup E \to T_E$ assigns a type to each vertex and to each edge. In accordance with Habel et al. [HHT96], the following has to hold for any subgraph $G_{sub} \subseteq G$: $G_{sub} = (V_{sub}, E_{sub}, src, trg, type)$ with $V_{sub} \subseteq V$, $E_{sub} \subseteq E$, and $\forall e \in E_{sub} : src(e), trg(e) \in V_{sub}$.

Moreover, Grunske et al. define *graph morphisms* between two graphs $G_1 = (V_1, E_1, src_1, trg_1, type_1)$ and $G_2 = (V_2, E_2, src_2, trg_2, type_2)$ as follows [GGZ+05]: A mapping $m : G_1 \to G_2$ consists of a pair of mappings (m_V, m_E), with $m_V : V_1 \to V_2$ and $m_E : E_1 \to E_2$. For these mappings, $\forall v \in V_1 : type_2(m_V(v)) = type_1(v)$ and $\forall e \in E_1 : type_2(m_E(e)) = type_1(e)$ hold. Further, they preserve sources and targets, i.e. $\forall e \in E_1 : m_V(src_1(e)) = src_2(m_E(e)) \wedge m_V(trg_1(e)) = trg_2(m_E(e))$ holds [CMR96]. If both mappings m_V and m_E are bijective, the mapping m is bijective, and the graphs G_1 and G_2 are considered *isomorphic* (denoted $G_1 \cong G_2$), i.e. that they have "equal structure but different naming" [Zün01].

While directed typed graphs can be used to describe structures or *snapshots* similar to object diagrams [Hec06], graph transformations can define how such structures may change. As defined by Rozenberg [Roz97], a graph transformation GT consists of two graphs: the *left-hand side* (LHS), which can be considered the application condition or precondition for the transformation, and the *right-hand side* (RHS), which defines the result or postcondition of the transformation. Additionally, Habel et al. [HHT96] introduced *negative* application conditions (NACs), which restrict the application of a rule to graphs, in which "some context specified in the [NAC] does not occur" [EHKZ05].

Grunske et al. [GGZ+05] explain the application of a graph transformation $GT = (LHS, RHS)$ on a directed type graph G as follows: As first step, a subgraph $G_{sub} \subseteq G$ for which an isomorphism $m : LHS \rightarrow G_{sub}$ exists needs to be identified. Only if such a subgraph exists, the precondition of the graph transformation is fulfilled. In such a case, the graph transformation GT is referred to as *applicable* on G. Secondly, all edges and vertices which are in $LHS \setminus RHS$ are removed from the matched subgraph $G_{sub} \in G$ [Hec06]. If this deletion removes a vertex v which is either the target or source of an edge e which is not deleted, e is referred to as a *dangling edge*. The removal of all remaining dangling edges as used here is referred to as the *single-pushout* approach (SPO, cf. [Roz97]). If the *double-pushout* approach (DPO) is used, on the other hand, a graph transformation which leaves dangling edges would not have been applicable in the first place [EEPT06]. As third step, all vertices and edges in $RHS \setminus LHS$ are added to the graph which results in the graph G'. In other words, the application of $GT = (LHS, RHS)$ on the subgraph G_{sub}, for which an isomorphic mapping m to LHS exists, consists of replacing G_{sub} with RHS which leads to G' (expressed as $G \overset{GT,m}{\Longrightarrow} G'$ [BH02]). Throughout this thesis, a shorthand version $G \overset{GT}{\longrightarrow} G'$ is used if either only one such matching subgraph exists or the resulting graphs are isomorphic independent of which mapping m is used to apply GT.

Combined Visualization in Story Patterns: As a kind of graph transformation, *story patterns* also consist of a left-hand side and a right-hand side [Zün01]. However, to emphasize the differences between these two sides and to visualize these differences more intuitively, story patterns encode the changes they affect as follows: all elements, i.e. vertices and edges, which are to be removed ($\in LHS \setminus RHS$) are displayed with a corresponding modifier ($--$ or ≪destroy≫, illustrated in red). Elements which are added ($\in RHS \setminus LHS$), on the other hand, have a green modifier ($++$ or ≪create≫) attached to them. Furthermore, any dangling edges which may remain are removed afterwards (i.e. *single-pushout* is used). Throughout this thesis, the terms story pattern and graph transformations will be used synonymously, since both can be considered equivalent except for the different visual representation of LHS and RHS which is defined for story patterns.

Additionally, each story pattern is directly assigned with an instantiated object it contains. This object is referred to as `this` object and the transformation is executed in its context [Zün01, GHS09]. Our approach exploits the existence of a `this` object in a story pattern SP_n to explicitly specify that the behavior represented by SP_n *belongs* to the `this` object (role v_1 in case of Figure 2.4).

Modifiable Natural Language Representation of Story Patterns (NL4SP): As directed typed graphs, the LHS and RHS of a story pattern can be considered equivalent to an UML object diagram. Based on this equivalence, Daniel Eichler implemented a transformation which generates a NL representation for story patterns in his master's

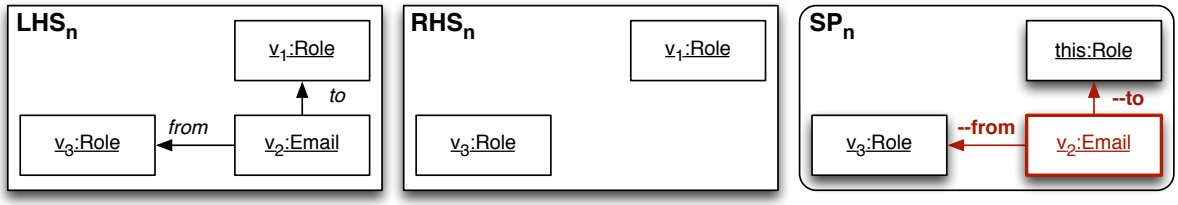


Figure 2.4.: A left-hand side LHS_n and a right-hand side RHS_n are visually combined in a story pattern SP_n (typed over T_V ={*Role, Email*} and T_E ={*from, to*})

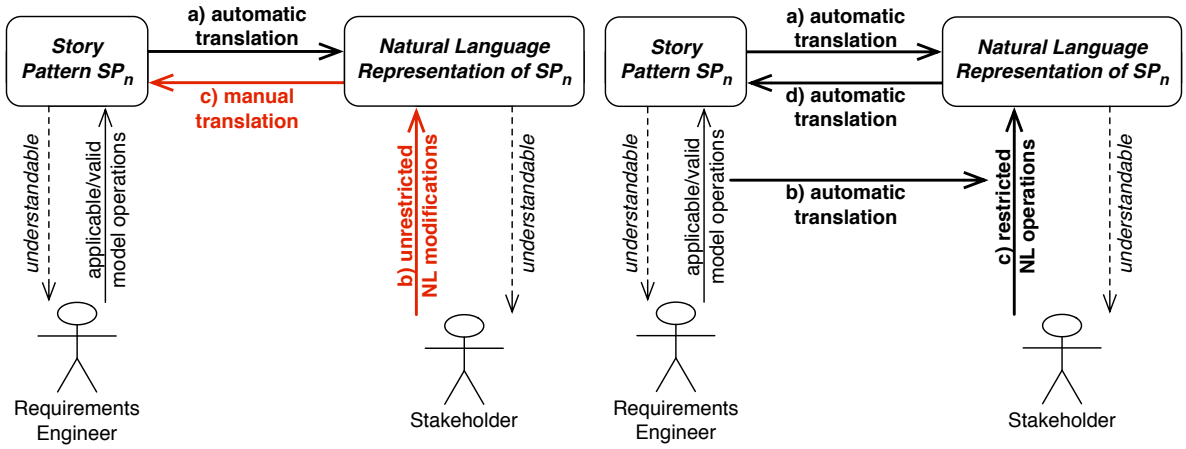
thesis [Eic12]. Vertices and edges in LHS, i.e. the preconditions of the story pattern, can be presented based on the types of the vertices and their edges. This can be done by enumerating the vertices and the edges between them, just as an object diagram or a class diagram as illustrated by Meziane et al. [MAA08]. For LHS_n in Figure 2.4, the result would be: v_2`(Email) is connected to` v_1`(Role) via edge` ***`to`***`.` v_2`(Email) is connected to` v_3`(Role) via edge` ***`from`***`.`

The changes that the story pattern encodes, however, are represented by the differences between LHS and RHS. These can also simply be enumerated. For Figure 2.4, the changes between LHS_n and RHS_n are: `Edge` ***`from`*** `between` v_3`(Role) and` v_2`(Email) is deleted. Edge` ***`to`*** `between` *this*`(Role) and` v_2`(Email) is deleted.` v_2`(Email) is deleted.`

The resulting NL representation becomes easier to understand, if the types in T_V and T_E are named intuitively. Additionally, naming the vertices intuitively as well (e.g. referring to v_3 as *Adam* instead), further improves the readability of the NL representation for stakeholders [GEHG13].

Presenting a NL representation of a story pattern enables stakeholders to understand the preconditions and changes encoded in the story pattern more intuitively. However, while such a representation can be generated automatically (*a* in Figure 2.5a), any stakeholder modifications of this representation (*b* in Figure 2.5a) cannot be automatically translated back into the underlying story pattern (*c* in Figure 2.5a). Instead, natural language processing techniques have to be applied to parse the modifications and try to (re)construct a meaningful model, which only works for simple sentence structures [ZD09].

Eichler's approach of providing *Natural Language for Story Patterns* (*NL4SP*) starts by generating a NL representation of a story pattern SP_n (*a* in Figure 2.5b). For each element in a story pattern, different valid model operations can be applied by a requirements engineer (*b* in Figure 2.5b): elements can be renamed, their types can be changed, or they may be deleted from or added to either side of the story pattern. In contrast to any unrestricted NL modifications, all of the above operations conform to the story patterns' metamodel, and, if applied, lead to a *consistently* modified story pattern SP'_n. Consequently, by translating only valid operations, mapping them on the corresponding (groups of) words in the NL representation of SP_n, and offering them to



(a) While a story pattern SP_n can be represented in NL (a), unrestricted stakeholder modifications (b) have to be mapped back manually (c)

(b) *NL4SP* provides applicable model operations in NL (b) so that stakeholder modifications are restricted (c) to operations which can automatically be translated back into SP_n (d)

Figure 2.5.: Stakeholders can modify story patterns more intuitively in NL – Eichler's NL4SP ensures that they do so consistently (adapted from [GEHG13])

a stakeholder (*c* in Figure 2.5b), the stakeholder's modifications are restricted to valid operations which can be translated back into the underlying story pattern (*d* in Figure 2.5b). Hence, stakeholders can modify a formal story pattern by manipulating (groups of) words within a more intuitive NL representation of the story pattern.

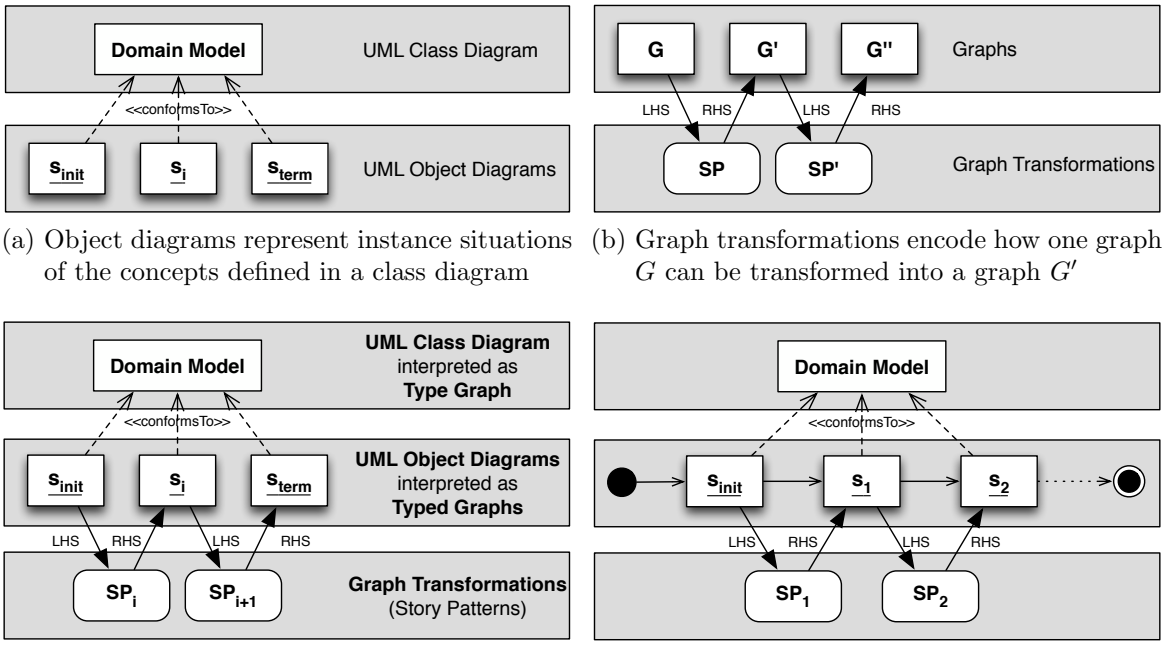
For the remainder of this thesis, it suffices to know that such an intuitive natural language interface which enables stakeholders to understand and manipulate story patterns is available (cf. [Eic12, GEHG13]).

3. Overall Approach

Scenarios in which multiple different stakeholders are involved who interact to achieve a common goal are referred to as *collaborative scenarios*. To specify and realize a software system which supports these stakeholders, requirements engineers have to investigate how these stakeholders interact. Only if the requirements engineers elicit how the stakeholders collaborate, can this software system fulfill the needs and requirements of those stakeholders. Thus, to support the elicitation and validation of knowledge which is fragmented and "often distributed in several involved people's minds" [BDML09], we provide a simulation approach which automatically elicits stakeholder activities based on how stakeholders play through their scenarios in groups or individually. Through an intuitive domain-specific visualization, stakeholders participate in a simulation of their scenario and can play in their usual activities and interactions. This behavior is recorded by the simulator and can be replayed in simulation sessions during which the stakeholder, who was originally recorded, cannot participate. By playing out these recorded fragments, the simulator provides feedback to the participating stakeholder on how other stakeholders might react on his or her actions. This allows requirements engineers to gather information that is distributed among different stakeholders in decoupled sessions with individual stakeholders.

Our approach aims at enabling individual stakeholders to validate collaborative scenarios they are involved in as well as to correct and complete these scenarios. This can be achieved by providing a *virtual prototype* of the scenario that individual stakeholders can directly experience, judge, and modify. The results, i.e. how a participating stakeholder is affected, are then intuitively animated relying on domain-specific metaphors. Our virtual prototyping approach relies on three different parts ($\mathcal{D}$, $\mathcal{S}$, $\mathcal{SP}$) required for the simulation and a domain-specific visualization which uses metaphors the stakeholders are familiar with. The three parts are as follows:

- a UML Class Diagram which is referred to as domain model $\mathcal{D}$ and which captures the concepts of the stakeholders' domain and how they interrelate
- a set $\mathcal{S}$ of UML Object Diagrams each of which consists of instances of domain concepts defined in $\mathcal{D}$ and represents a situation which may occur in the stakeholders' domain
- a set $\mathcal{SP}$ of story patterns which capture behavior observed from stakeholders



(a) Object diagrams represent instance situations of the concepts defined in a class diagram

(b) Graph transformations encode how one graph G can be transformed into a graph G'

(c) By interpreting object diagrams as graphs conforming to the domain model, graph transformations define how to get from one object diagram to the next

(d) In our state-based simulation, object diagrams represent scenario states; graph transformations describe observed state changes and can be executed to simulate these changes

Figure 3.1.: Concepts, states, and behavioral models in our approach

As pointed out in Section 2.2, class diagrams prescribe all valid states which corresponding object diagrams may represent (Figure 3.1a) and graph transformations describe the differences between pairs of graphs (Figure 3.1b). Arijo et al. [AHTG11] as well as Baresi and Heckel [BH02] pointed out that an object diagram conforming to a class diagram is equivalent to a typed graph conforming to a type graph [CMR96]. Consequently, instantiated concepts in an object diagram correspond to nodes in a graph whose type is defined in a type graph similar to $\mathcal{D}$. Furthermore, associations between instances within the object diagram are equivalent to directed typed edges between nodes in a typed graph. By utilizing this equivalence, we are able to define concepts in $\mathcal{D}$, instantiate these in object diagrams $\mathcal{S}$ which are interpreted as graphs. This, in turn, allows the usage of story patterns $\mathcal{SP}$ to describe transitions between these graphs (Figure 3.1c). Additionally, since the concepts in $\mathcal{D}$ represent the stakeholders' domain, the object diagrams represent all possible situations the domain may be in, as per definition. Thus, provided $\mathcal{D}$ defines concepts associated with the scenarios the stakeholders are involved in, such object diagrams can be used to specify distinct situations which may occur at specific points in time during the execution of these scenarios. In other words, all possible object diagrams which conform to $\mathcal{D}$ are considered *states* the stakeholder scenarios may potentially be in. This combination of scenario states, which are based on

domain concepts in $\mathcal{D}$, and story patterns which specify transformations between these states allows us to simulate these scenarios (Figure 3.1d).

The possibility to execute and simulate formal models provides requirements engineers with new insights concerning aspects such as the overall consistency of the gathered behavioral models. Stakeholders, on the other hand, do not directly benefit from such capabilities, since they are usually not trained to understand the results of such simulations. However, by employing an intuitive representation of the current state of the simulation, stakeholders may directly experience this simulation. This enables them to comment on the current state of the simulation which aims to represent the scenarios they are involved in and, thus, indirectly the behavioral models which change the state of the simulation (Figure 3.2).

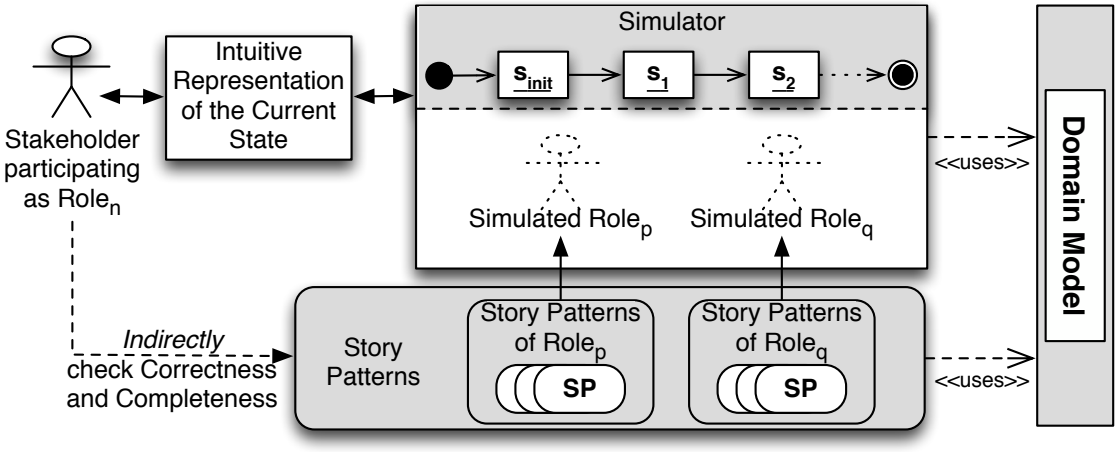


Figure 3.2.: Using an intuitive representation of situations in a simulated scenario, stakeholders experience how simulated activities of other roles affect them

In our approach, each scenario involving multiple collaborating stakeholders can be considered a flow of activities of stakeholders and software systems between a sequence of situations in which the stakeholders involved in the scenario find themselves. For instance, sending an email with a document attached to it or handing over a contract which needs to be signed are activities leading to new situations (in these cases: somebody having access to a copy of the document and somebody else being able to sign the contract, resp.). By describing such situations using the language established by $\mathcal{D}$, the structure of those states $\mathcal{S}$ between the start and the end of the scenario can be captured, modified, and behavioral specifications $\mathcal{SP}$ can be derived from and executed on top of these states.

The extendable domain model $\mathcal{D}$, its instantiations in distinct states in $\mathcal{S}$, and behavioral specifications $\mathcal{SP}$ are discussed in Section 3.1, which is partially based on our publication [GGS10b]. Then, Section 3.2 (partially based on [GEGS10]) illustrates how these states of collaborative scenarios can be visualized based on domain-specific

metaphors, i.e. in a way that is intuitively understandable for stakeholders involved in these scenarios.

3.1. Modeling Collaborative Scenarios

Based on Davis et al.'s analysis of more than 4 000 papers related to *requirements engineering* which were published during 50 years of research [DHD+07], they reported that only few of these papers discuss the subjective *perception of reality* which also affects requirements engineers [DN07]. However, only if the stakeholders' domain is investigated and modeled accordingly, can requirements be built upon the domain model, as Jackson pointed out [Jac00]. He further argues that a set of requirements can only be *correct* with respect to the domain model $\mathcal{D}$ for which they are defined. Consequently, a domain model $\mathcal{D}$ has to contain the concepts required to define the requirements capturing the stakeholders' needs. To provide a solid foundation to work with, we introduced an extendable domain model referred to as $\mathcal{D}_{basic}$, i.e. *basic* domain model (Figure 3.3, cf. [GGS10b]). $\mathcal{D}_{basic}$ already contains the concepts necessary for describing situations as they occur in collaborative scenarios taking place in workplace environments. For instance, it can be expressed that an *email* was sent *from* a *role to* another *role*. In our approach, $\mathcal{D}_{basic}$ is used as a language for describing such situations and states which, in turn, *conform to* $\mathcal{D}_{basic}$. Hence, the domain model can be considered as *type model* [Küh06]. Still, by extending this domain model and, thereby, enhancing the language that it defines, requirements engineers can describe specific situations within particular projects in even more detail.

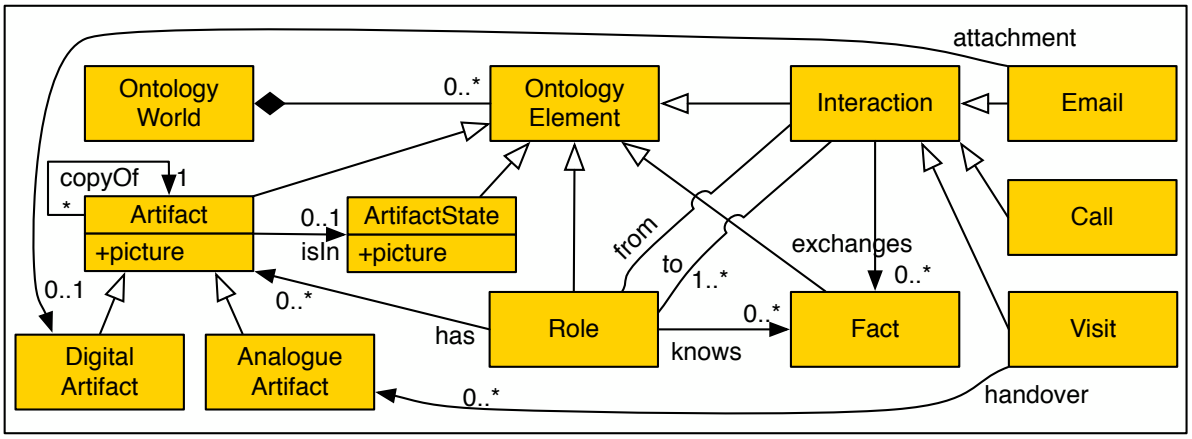


Figure 3.3.: Projects start from this basic domain model $\mathcal{D}_{basic}$ by defining subclasses suitable for the context of the project (adapted from [GGS10b])

3.1.1. Refining the Basic Domain Model to Describe Scenario States

At the beginning, when the collaborations between different stakeholders within their scenarios have to be elicited, the basic domain model $\mathcal{D}_{basic}$ establishes a vocabulary that can be used as a starting point to describing states and, thus, the stakeholders' behavior that changes these states. This basic domain model illustrated in Figure 3.3 structures the general domain of *collaborative scenarios involving multiple stakeholders*. While stakeholders or stakeholder groups can be characterized by *roles*, how they communicate can be described by different kinds of *interaction*. Additionally, the artifacts the stakeholders work with and the knowledge they require to do so can be described by *artifact* and *fact*, respectively.

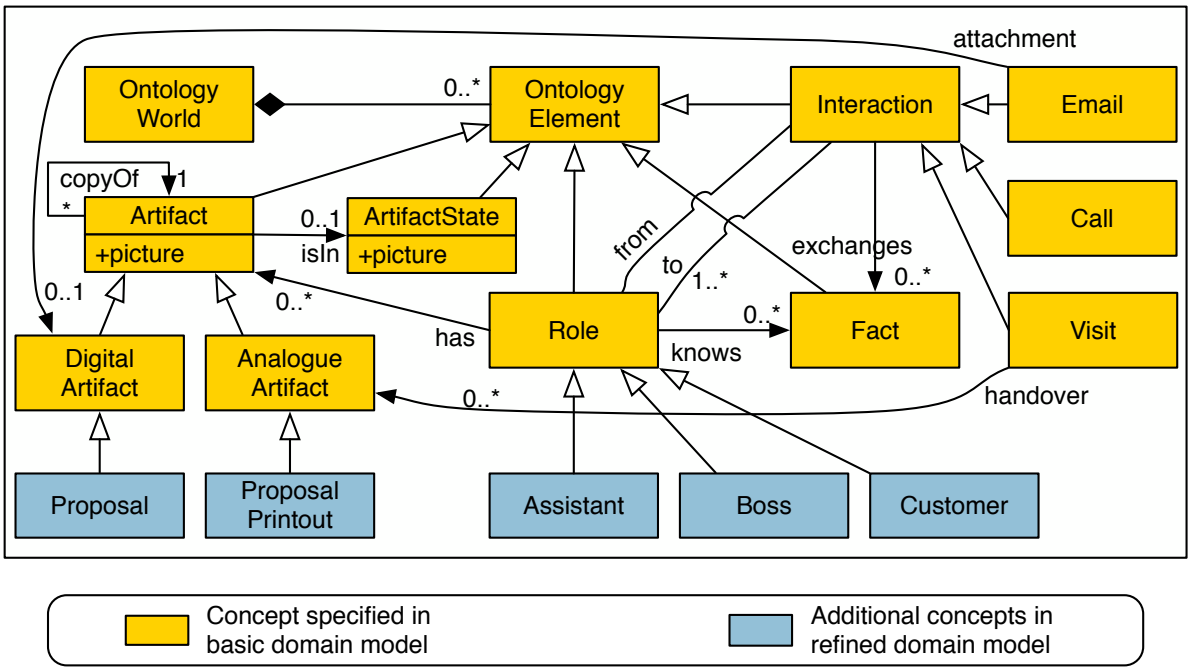


Figure 3.4.: For each domain, roles and artifacts are identified and defined in $\mathcal{D}_{specific}$

Further, to use, extend, and refine the rather generic concepts of $\mathcal{D}_{basic}$, a second domain model $\mathcal{D}_{specific}$ is used to capture the subclasses that are defined by the requirements engineer. By adding or refining concepts to $\mathcal{D}_{specific}$, the language that is defined by $\mathcal{D}$, which combines $\mathcal{D}_{basic}$ and $\mathcal{D}_{specific}$ and, therefore, consists of all concepts, becomes more expressive. Similarly to how cars are undistinguishable without concepts for describing their specific parts, their brands, or their color, the same goes for describing collaborative scenarios; talking about generic documents being passed around by generic roles is useless, until specific manifestations can be described, distinguished, and later on recognized by stakeholders. A simple example of such a scenario-specific extension

of $\mathcal{D}$ is shown in Figure 3.4. Concepts defined in $\mathcal{D}_{basic}$ are illustrated in orange, while concepts captured in $\mathcal{D}_{specific}$ are displayed in blue. For this example, three different roles were already identified in initial interviews: `Assistant`, `Boss`, and `Customer`.

Starting with $\mathcal{D}_{basic}$ as shown in Figure 3.3, project specific subclasses of `Role`, `Fact`, `Artifact`, or even `Interaction` can be defined in $\mathcal{D}_{specific}$ as illustrated in Figure 3.4. Then, specific situations or states can be described in conformance to $\mathcal{D}$. The specification of an email which is sent from one person to others (with each recipient receiving an individual copy of the email) is necessary, since it can be considered as a communication primitive [GH06]. The states before and after such an email was sent are illustrated in Figure 3.5. Corresponding to the domain-specific refinement of the domain model, the refined roles and artifacts are the ones which are interacting and modified along the way, respectively.

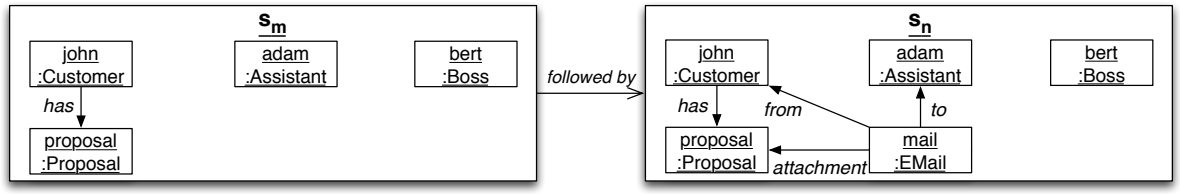


Figure 3.5.: Abridged states representing the situations before (s_m) and after (s_n) the *assistant* Adam receives a proposal from the *customer* John

As soon as new collaborative scenarios are to be investigated in a new project, $\mathcal{D}_{basic}$ (possibly including suitable and approved extensions as we described in [GGS10b]) is used as the initial domain model and can iteratively be extended by a new $\mathcal{D}_{specific}$. Potentially, each refinement enhances the language established through the domain model in such a way that it can then be used to describe states which were not expressible with prior versions of $\mathcal{D}$.

Since we are interested in eliciting specific collaborative scenarios (e.g., signing a business proposal), any potential impacts of other instances of the same scenario (e.g., multiple instances of the proposal existing within the same state) or parallel executions of other scenarios should be excluded. Thus, our approach works under two specific assumptions:

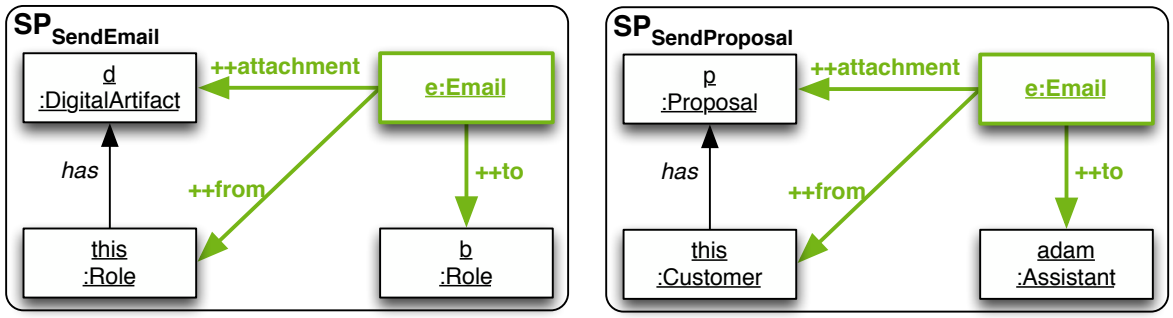
1) Similar to the categorization introduced in Alexander's *stakeholder onion* [Ale05], different stakeholders involved in a collaborative scenario can be consolidated into specific *roles*, i.e. subclasses of `Role`. For instance, while there might be only one stakeholder who everybody refers to as *"boss"*, there may be multiple stakeholders who work for this `Boss` as `Assistant`. Consequently, any of these assistants does have the necessary knowledge to enact the *role of an assistant* during the enactment of the scenario. Therefore, only one instance of each specific subclass of `Role` defined in $\mathcal{D}$ is required for the simulation of the scenario.

2) Our approach distinguishes between different artifacts by introducing refinements thereof, which reflect the distinction from a stakeholder's point of view. Such refinements enable the requirements engineers to describe scenario states in a level of detail which makes the different artifacts distinguishable for stakeholders. Consequently, just as their real-life counterparts, it is always possible to introduce a refined concept in $\mathcal{D}$ for each domain concept that needs to be distinguishable.

While these assumptions require a sufficiently expressive domain model $\mathcal{D}$, they do not restrict the applicability of our approach.

3.1.2. Story Patterns based on Scenario States

The domain model $\mathcal{D}$ can be considered the type graph for graphs which, in turn, represent states that describe situations occurring within the collaborative scenarios under investigation. From a subsequent pair of such situations, the changes between them can be expressed using graph transformations, i.e. story patterns. An example that is part of a scenario concerned with *signing a business proposal* is illustrated in Figure 3.5. As a customer, John would say: "`After having prepared a proposal, I send it to the assistant via email`." From an external point of view, the observation is limited to the prior and subsequent situation (s_m and s_n, resp.) as well as to which one of the interacting stakeholders is *actively* changing the situation. Still, objectively we observe the situations illustrated in Figure 3.5 as follows: "`When the customer` (the *active* role) `has a proposal, the assistant then receives an email sent by the customer containing this proposal`." This apparent difference between these two states can be described using story patterns: as soon as the prior situation is observed ($LHS = s_m$), customer can conduct an activity which leads to the later situation ($RHS = s_n$). Moreover, observed changes between subsequent states can be defined as a story patterns which are then assigned as executable behavior to one of the participating roles (indicated by naming the corresponding instance of role `this`). By restricting `this` objects to be instances of *roles*, the behavior captured in story patterns is bound to and *belongs* to the corresponding group of stakeholders. An example of generic behavior is shown in Figure 3.6a: since the applicability of this story pattern is not further restricted to specific roles, any role can send a digital artifact to any other role via email. However, to capture a distinct interaction that can be observed within any of the scenarios under investigation, we simply have to refine this specification to be applicable only for the corresponding situations which usually invoke this behavior. As Figure 3.6b shows, it is possible to define which `sender` is supposed to send which `attachment` (which needs to be a digital artifact, as specified in $\mathcal{D}$) to which `receiver`. This new specification relies on refinements defined in the project-specific parts of $\mathcal{D}$. Consequently, specifications refining generic concepts can be used to specify distinct stakeholder actions or interactions with other (potentially simulated) roles.



(a) While any (subclass of) `Role` can send an `Email` with attached `DigitalArtifacts`, ...

(b) ... only a `Customer` may execute the story pattern for sending a `Proposal`

Figure 3.6.: The story pattern $SP_{SendProposal}$ refines the generic version $SP_{SendEMail}$ (adapted from [GGS10b])

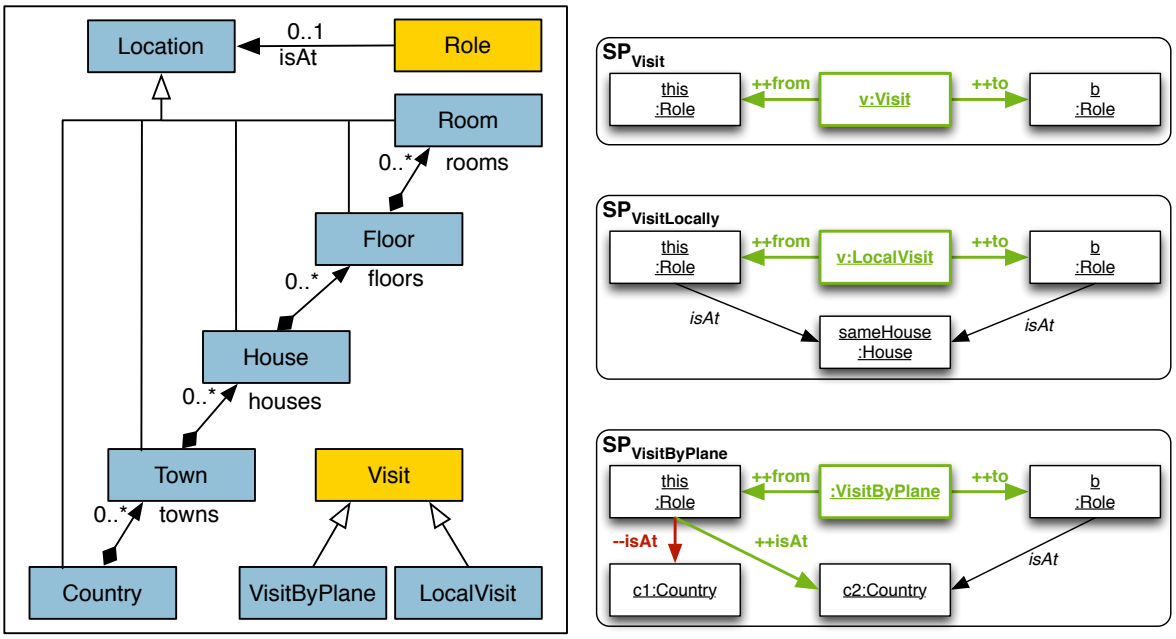
3.1.3. Introducing new Concepts into the Domain Model

By carefully studying Figure 3.3, one might find several gaps which inhibit the usage of the generic $\mathcal{D}_{basic}$ for modeling most scenarios. While $\mathcal{D}_{basic}$ easily affords the definition of specific roles, it does not include concepts necessary for, e.g., describing scenarios involving traveling stakeholders in globally distributed companies. In the following, we demonstrate one of multiple possible solutions for modeling this specific challenge to illustrate how the domain model $\mathcal{D}$, which our approach employs, bridges such gaps.

$\mathcal{D}_{basic}$ already contains the concept of one role visiting another one (`Visit`). While one is usually able to visit coworkers working on the same floor, the situation is different for coworkers in other towns, countries, or even on other continents. To reflect this distinction in the domain model, it is necessary to include information about the *location* of *roles* into $\mathcal{D}$. One possible solution is presented in Figure 3.7a. Each role can be modeled to be in a `Room` which can be assigned to a `Floor`, `House`, `Town`, or `Country`. Depending on the size of the company, it might suffice to include rooms on floors within the same building. Then, *visits* are still possible without considering the time it takes to reach the target or the resources consumed to get there. However, if somebody working in Germany needs to visit a colleague in Canada, a visit can take several days and cost thousands of dollars. Since this distinction between different types of visits relies on locations, a `Location` concept must be modeled within the project-specific domain model first (cf. Figure 3.7a). Then, refinements of `Visit` can be described as illustrated in Figure 3.7b.

3.1.4. Handling Domain Concept Modifications

Usually, modeling tools rely on a predefined metamodel and the language this metamodel establishes. Concepts which were not anticipated by the tool provider are not covered in



(a) Concepts for `Location` and refinements of `Visit` are added to $\mathcal{D}$ to model collaborative scenarios within globally distributed companies

(b) After concepts related to `Location` have been added to $\mathcal{D}$, different subtypes of `Visit` can be distinguished and specified in story patterns

Figure 3.7.: Extending the concepts in the domain model $\mathcal{D}$ allows to describe more complex situations and, hence, behavior (adapted from [GGS10b])

such a metamodel and, hence, cannot be modeled. As a result, predefined metamodels restrict the requirements engineers and are generally considered rather inconvenient due to this inflexibility. However, an inflexible metamodel avoids the problem of models becoming invalid due to changes in their metamodel. The same problem applies for the models that have to conform to $\mathcal{D}$: While additional concepts enrich the language that is established by $\mathcal{D}$ as discussed in Section 3.1.3, modifications or the removal of concepts invalidates all models referring to the outdated concepts. As a consequence, these models do not *conform* to the updated version of $\mathcal{D}$. To solve this problem, we propose to circumvent invalid references between these states and story patterns to concepts in $\mathcal{D}$ by redirecting them to the corresponding superclass in $\mathcal{D}$ or even $\mathcal{D}_{basic}$. For instance, if the concept `Customer` in $\mathcal{D}$ (cf. Figure 3.4) is deemed inappropriate and marked for deletion, the story pattern $SP_{SendProposal}$ in Figure 3.6b would become invalid, since it contains an instance of and, thus, references the concept `Customer`. In the worst case, this means losing the specified behavior completely. While the interaction might not be valid for a `Customer`, the observation that it took place still remains valid. By redirecting this reference to `Role`, the story pattern remains valid and executable. However, information is lost, since the story pattern's precondition is less restrictive. Instead of capturing a potentially unique interaction between a customer and an assistant, the resulting, more

generic story pattern implies that *any* role can send an email with an attached proposal to the assistant. The same substitution is possible for almost all elements of a domain model, as long as there is a valid superclass, i.e. more generic concept, in $\mathcal{D}_{basic}$ (in our case `OntologyElement`, cf. Figure 3.3).

Since all relevant changes originate in the domain model $\mathcal{D}$, this category of changes is considered *metamodel-specific* [HBJ09] and can be applied to all instances of `OntologyElement`. Consequently, all instances of a modified concept which are part of a state s conforming to $\mathcal{D}$ or that are part of either the LHS or RHS of a story pattern $SP \in \mathcal{SP}$ can be adapted correspondingly. It has to be ensured that all affected models in $\mathcal{S}$ and $\mathcal{SP}$ are always updated accordingly, i.e. only contain valid references to the concepts defined in $\mathcal{D}$. The simple replacement illustrated in this section suffices in most cases – since approaches for this category of problems (e.g., *COPE* for the "coupled evolution of metamodels and models" [HBJ09])[1] already exist, more complex replacements are out of the scope of this thesis.

3.1.5. Use Cases for Domain Model, States, and Story Patterns

By enabling requirements engineers to modify the domain model $\mathcal{D}$, we avoid the drawbacks of an premature commitment to a specific formal tool or metamodel [OBS+10]. At the same time, the distinction between generic concepts in $\mathcal{D}_{basic}$ and specific concepts added later on to $\mathcal{D}$ guarantees domain-specific semantics [GGLS11]. Consequently, the use cases concerning the domain model $\mathcal{D}$ that are relevant for the requirement engineers are illustrated in Figure 3.8. Initially, the *elicitation* together with stakeholders and the subsequent *specification* of concepts during which $\mathcal{D}$ is defined similar to a glossary (D1 and D2, respectively) takes place. Afterwards, to ensure that a common understanding is established between the stakeholders and requirements engineers, $\mathcal{D}$ can be *validated* (D3), e.g. by producing an easy to understand natural language version of the modeled relationships between the concepts of the domain [MAA08]. Incorrect assumptions of the requirements engineers may manifest in erroneous concepts in the domain model. Especially in a specific situation during a simulation, a stakeholder can point out that, e.g., an additional distinction between different types of proposals is required or that a specific role such as `Customer` can be removed as mentioned above. Then, the domain model can be *adapted* based on stakeholder feedback (D4), if deemed necessary. Of course, adapting $\mathcal{D}$ includes modifications such as removal and addition of concepts that might invalidate states or even story patterns. Thus, invalid references from states or story patterns to $\mathcal{D}$ have to be checked and potentially adapted as well (D5).

Initial or terminal states, on the other hand, can be described by stakeholders (S1 in Figure 3.9). This enables requirements engineers to use this description to create a state specification relying on the vocabulary established by $\mathcal{D}$ (S2) – this may even be achieved by the stakeholders directly, provided they can use an intuitive editor for doing

[1] $\mathcal{D}$ is employed in such a way that it is equivalent to a metamodel for our states and story patterns.

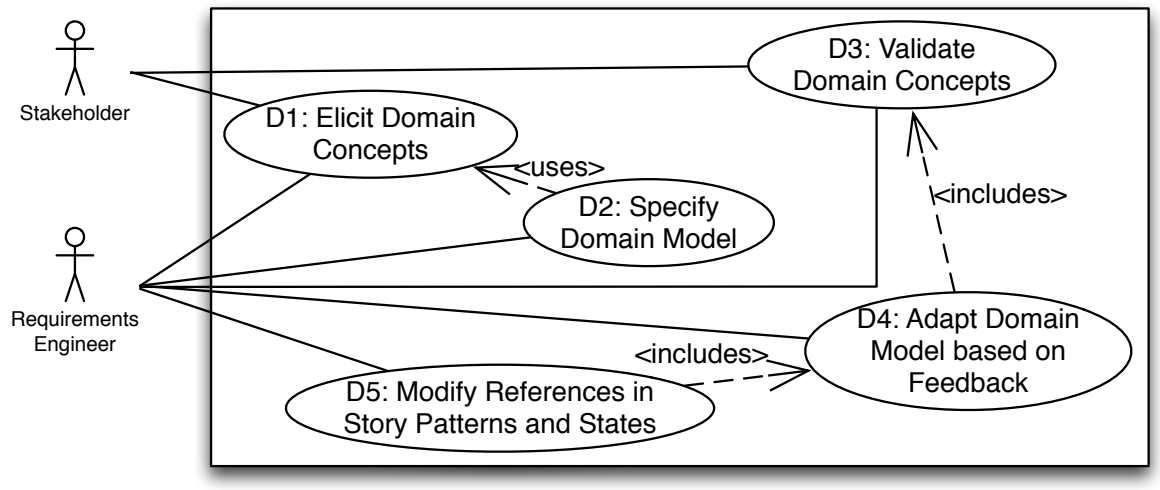


Figure 3.8.: Use cases related to the domain model $\mathcal{D}$

so (cf. Section 2.5). After a state has been defined, a requirements engineer may either modify it based on stakeholder feedback (S4) or even remove it (S3).

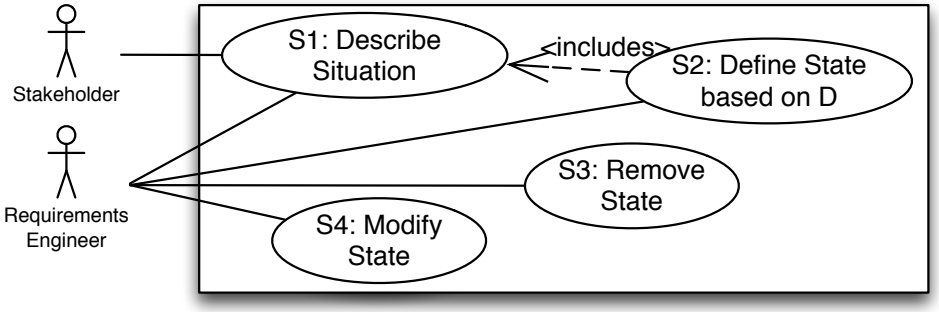


Figure 3.9.: Use cases related to distinct states in $\mathcal{S}$ which are based on $\mathcal{D}$

We chose story patterns to formally capture stakeholder actions and interactions. Still, stakeholders need to be able to experience *a)* a state and *b)* its changes which reflect activities of other stakeholders or software systems. By recognizing such a state and changes within it, they are able to evaluate whether what they see matches their experience or expectations in the respective state during a simulation. Consequently, such a representation enables stakeholders to point out factual errors directly in the state that is being visualized as well as indirectly in the story pattern(s) that led to this state. Thus, by collecting story patterns and validating them with stakeholders, they can identify errors and report them to requirements engineers (SP1 in Figure 3.10). Since each piece of the stakeholder's story depends on the precondition and the resulting

postcondition, either one of these conditions might have to be adapted (SP2 and SP3, respectively) or the story pattern is removed (SP4).

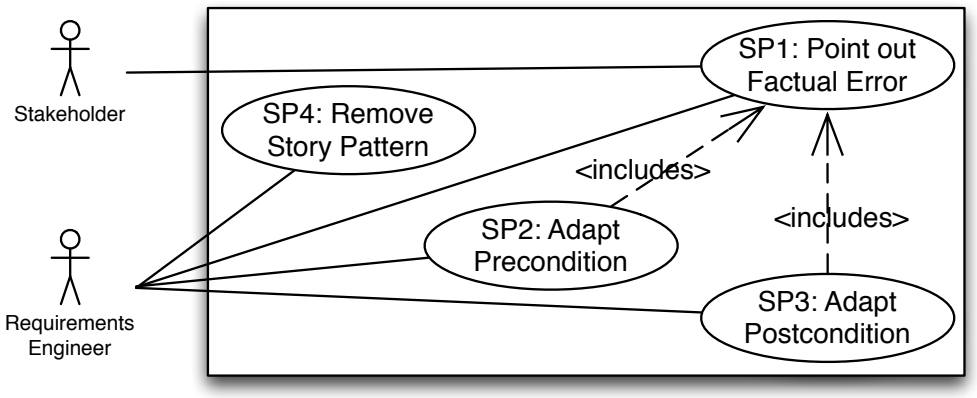


Figure 3.10.: Use cases related to the story patterns in $\mathcal{SP}$

3.2. From Formal Models to Virtual Prototypes

To effectively prototype collaborative scenarios with stakeholders, the models described in Section 3.1 are not sufficient. As Schrage [Sch04] argues, "we model, prototype, and simulate software with clients, not for them". Consequently, to validate and elicit their requirements, they need to be able to understand, judge, and complement what is presented to them. What makes *tangible* prototypes particularly effective is the stakeholders' ability to recognize whether a prototype is a sketch of a suitable solution that fulfills their needs.

Formal models, on the other hand, are used quite commonly in engineering disciplines. Their elements carry semantic and syntactic information that are not obvious to most stakeholders. Consequently, such models are either used to efficiently communicate knowledge between trained professionals or to benefit from automation capabilities provided by these models. The main differences between these two approaches of representing ideas, however, is whether a stakeholder has to be trained to understand the representation of the information. This section discusses how intuitive visualizations of states conforming to $\mathcal{D}$ and story patterns in $\mathcal{SP}$ executed on them can be used to *virtually* prototype the collaborative scenarios under investigation.

3.2.1. Prerequisites of Representations

To communicate, iterate, and validate an idea with stakeholders or other engineers, the idea has to be externalized and represented in order to be shared. The more complex or abstract the idea, the more difficult it is to create a representation that stakeholders

can understand intuitively, i.e. that is considered "tangible". Furthermore, the choice of the representation influences what can be modeled, recognized, and which operations can be executed. Moreover, this choice also determines which background any person judging it requires in order to be able to work with it.

As an example, we may consider the following two different representations of numbers. Early in school, most people learn to multiply Arabic numerals such as 177×23 as illustrated in Table 3.1. However, if the multipliers are represented using Roman numerals, i.e. $CLXXVII \times XXIII$,[2] only few people are capable of doing so without transforming the numerals into the Arabic format first. Therefore, which operations (in this case *multiplication*) a person is capable of executing on a specific representation of information depends on *a)* how it is represented (Arabic or Roman numerals) and *b)* whether this person was trained to work with or understand this specific representation.

177	×	23	= 4071
		354_	
+		531	
		4071	

Table 3.1.: Multiplication of Arabic numerals as it is taught in school

Tangible Prototypes as Representations without Prerequisites: Lim et al. [LST08] argue that prototypes are manifestations of design ideas that "filter the qualities in which the designer is interested without distorting the understanding of the whole". Consequently, stakeholders need to able to recognize the properties of the prototype and interpret which of those are relevant to provide suitable feedback about them. Tangible prototypes can be touched, experienced, and evaluated without any specific prerequisites (cf. Figure 3.11a). The only required knowledge imposed by such a tangible prototype is that the stakeholders have to be familiar with the domain. By matching their interpretation of the prototype with their perception of the problem they want to have solved, all stakeholders can provide feedback.

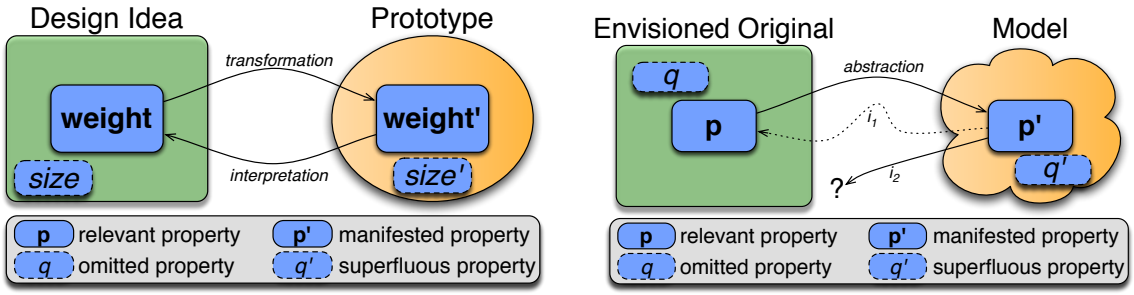
Prerequisites of Models: Models, on the other hand, are commonly used to describe inherently intangible concepts such as software systems. By modeling either an existing or an envisioned original, engineers can externalize and share their thoughts with other engineers. According to Stachowiak [Sta73], models are abstract representations of existing or envisioned originals, i.e. they have a *point of reference*. To be feasible, not all properties of this original are represented in the model. Without such a reduction, a model would be impractically complex. Pohl [Poh10] defines a model as "an abstract

[2] This example and an illustration of the procedure of how to multiply Roman numerals can be found at `http://www.phy6.org/outreach/edu/roman.htm` (accessed June 2013)

representation of [aspects of the existing or conceived reality under consideration] created for a specific purpose". By emphasizing certain aspects, the overall complexity decreases and can be dealt with. As indicated by Rothenberg [Rot89], modeling is a way of dealing with things or situations that are too costly to deal with directly. In the case of the construction of bridges, the engineers want to know whether the planned bridge withstands the expected load before they build it. To achieve this, they model the bridge and analyze these models to predict or verify the physical properties of the bridge.

In several domains dealing with abstract and intangible concepts, e.g., software engineering or molecular chemistry, domain-specific models enable experts to externalize and communicate their knowledge or ideas with other experts. In such a setting, the involved experts are from the same domain and, therefore, will likely share the same interpretation (e.g., i_1 in Figure 3.11b) of the model. Having such a common understanding of what is modeled allows them to effectively communicate concerning the existing or envisioned original. If, however, stakeholders are involved as well (e.g. during validations), such a common interpretation can usually only be achieved if the representation of the model mimics the original as close as possible. Unless an expert explains these models, the underlying modeling language, its elements, and their symbolic meaning to the stakeholders, each stakeholder would interpret the model differently, if not incorrectly (e.g., i_2 in Figure 3.11b).

The creation of a prototype to externalize a design idea can be considered equivalent to the creation of a model prescribing an envisioned original. In both cases, knowledge is externalized to be communicated with other persons, i.e., other experts or stakeholders. Additionally, the created representations *by definition* do not reflect all properties – instead, specific properties are emphasized and others ignored. For instance, while transformations from UML class diagrams to natural language [MAA08] afford stake-



(a) Relevant properties of design ideas can be manifested tangibly in prototypes

(b) An (envisioned) original that is externalized as a model introduces prerequisites for its correct interpretation i_1

Figure 3.11.: Feedback can be fostered by externalizing visions or ideas

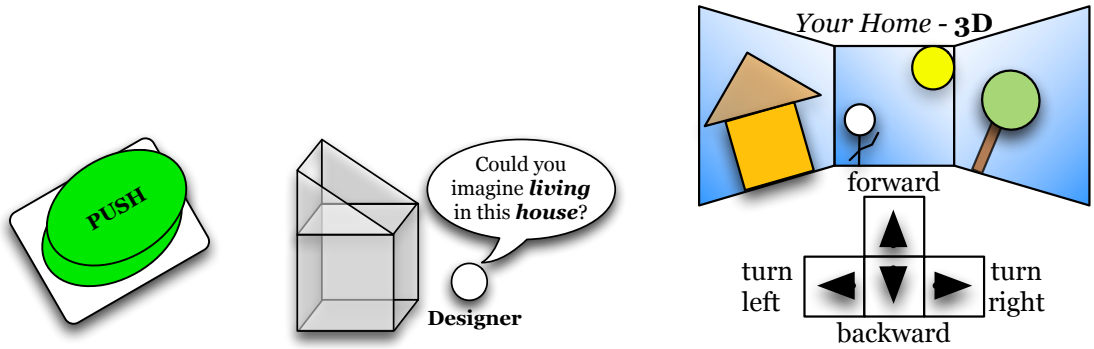
holders to understand and validate this informal representation of these models, other transformations support specific operations by focusing on a specific aspect such as *causal dependencies* to simplify the identification of inconsistencies [LSW08]. As mentioned before, representations impose prerequisites to be interpreted correctly. Thus, the main difference between prototyping and modeling is the fact that prototypes usually build upon the stakeholders' domain knowledge, while models require stakeholders to be trained to understand the employed modeling language and its notation [DS99]. Otherwise, models may only be used to communicate and share ideas among trained experts. Therefore, as externalization of knowledge, concepts, or ideas, prototyping can be considered as a specialized form of modeling which sacrifices unambiguity and expressiveness (for trained experts) for understandability for stakeholders.

3.2.2. Affordance Options

Only if all stakeholders (and modelers) recognize and interpret a prototype's property p' as the manifestation of the design idea's property p, can they share the same understanding of this property (as illustrated in Figure 3.11) and iterate the underlying idea. Affordances, as defined by Gaver [Gav91], are properties of the world that are compatible with and relevant for people's interactions. This definition of affordances is suitable to describe the stakeholders' ability to recognize and interpret a property represented in a model or prototype correctly. Moreover, Gaver argues that whether an affordance is perceived depends, among other things, on the observers' experience [Gav91]. For instance, while an architect is able to envision herself walking through the building she sees sketched on an architectural drawing, somebody unfamiliar with such models is not able to do so – let alone find the entrance of the envisioned building. This dependency on each stakeholder's individual experience or expertise leads to three different choices to achieve *tangibility* for prototypes:

- The first option is to build prototypes that offer an affordance by relying on concepts that everybody is familiar with (cf. Figure 3.12a). Everybody who ever used a labeled button would recognize the associated affordances in other contexts as well. While it is desirable to create affordances in such a way, it is hardly possible for inherently abstract concepts the stakeholders are not familiar with.

- The second option would be to teach stakeholders how to perceive affordances which would otherwise be new, unfamiliar, and, thus, hidden for them. This can be achieved by, e.g., showing or teaching them how they can interact with a prototype or how they are supposed to interpret a specific representation. Consequently, the observers gain new experiences which allow them to recognize such affordances later on again (cf. Figure 3.12b). However, a prototype that must initially be explained to all stakeholders to enable them to evaluate the embodied concepts is infeasible for projects which involve many stakeholders since each one of them



(a) Creating obvious affordances (b) Teaching to perceive affordances (c) Present affordances within the stakeholders' domain of expertise

Figure 3.12.: There are different possibilities of creating affordances that can be perceived intuitively

would have to be taught beforehand. Further, the explanations may change the stakeholders' interactions with the prototype which may affect the authenticity of their reactions and, hence, their feedback.

- A third option, suitable especially for inherently abstract concepts, is to provide stakeholders with a *common interpretation* of the otherwise potentially incomprehensible prototype (cf. i_{common} in Figure 3.13). While this is a common practice for requirements engineers who, e.g., either annotate their formal models or even keep redundant natural language representations [ARE96], doing so for many iterations is infeasible due to the high costs of creating such representations manually. To create such a representation feasibly, an automatic transformation of the relevant properties of ideas is necessary. This transformation must not require specific experiences to perceive the manifested concepts and to evaluate the underlying design idea (cf. Figure 3.12c). This is possible by explicitly relying on the commonly shared domain of expertise of the stakeholders (captured in $\mathcal{D}$), which is referenced in the requirements engineers' models (in our case: $\mathcal{SP}$ and all states conforming to $\mathcal{D}$).

Consequently, the suitability of a prototype for a specific stakeholder or a group of stakeholders depends on the prototype's prerequisites concerning the background of the person that needs to judge the prototype. Since the models that engineers create either *describe* details, relations between them and requirements from within this domain or *prescribe* an idea how specific stakeholder needs can be fulfilled, the engineers can always rely on the domain-specific knowledge of the stakeholders. The engineers can build upon this knowledge by providing representations that all stakeholders can uniformly interpret.

3.2.3. Virtual Prototypes

As illustrated in Figure 3.13, a model can be experienced similar to a tangible prototype, if the stakeholders are enabled to understand its content and, in case of dynamic behavior, how it affects them – ideally within their domain of expertise. Thus, the behavioral specifications collected from any of the involved stakeholders can be prototyped directly by enabling stakeholders to experience how they are affected through story patterns of other stakeholders. Then, the stakeholders can comment on what these models represent without being hindered by the story patterns' idiosyncratic formal representation. Consequently, our solution maps the content of these complex models back into the domain of expertise of the stakeholders to close the gap between behavioral specifications on the one side and tangible prototypes on the other side. By executing a story pattern and automatically mapping its effects into the stakeholder domain, stakeholders are enabled to understand the behavior specified in the story pattern. Such effects are usually based on interactions with stakeholders, actions like prompting stakeholders or reacting to stimuli from them. Creating such effects through simulating and visualizing those using domain-specific metaphors provides a common interpretation i_{common} which can be perceived intuitively and uniformly as long as it is solely based on the stakeholders' domain (cf. Figure 3.13). Of course, for complex systems supporting collaborative scenarios involving different groups of stakeholders with different backgrounds, each group of stakeholders might need a different visualization of the model within their individual domain of expertise, similar to the effects observed by Sellen et al. [SML+09]. Similarly to tangible prototypes, different aspects can be emphasized or omitted.

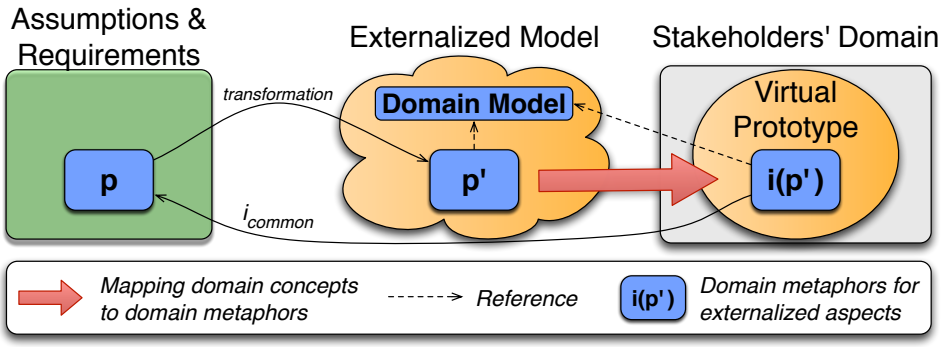
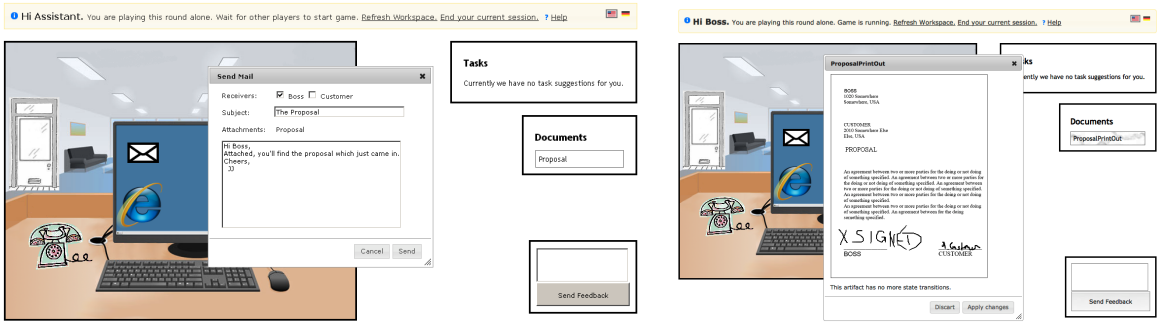


Figure 3.13.: An externalized model which imposes prerequisites on understanding needs to be transported back into the stakeholders' domain of expertise

Based on the domain model $\mathcal{D}$, specific states $\mathcal{S}$ of the collaborative scenarios, and the collected behavioral specification $\mathcal{SP}$, it is possible to provide a framework to create *virtual* prototypes, as opposed to *tangible* ones. By consistently employing domain-specific metaphors that are intuitively understandable for the stakeholders and that they are familiar with, it can be ensured that most if not all of them share a common interpretation of what is presented to them (cf. $\mathbf{TG}_{2A}$ in Section 1.2). Thus, a virtual

prototype VP consists of $\mathcal{D}$, $\mathcal{S}$, $\mathcal{SP}$, and a domain mapping which defines how concepts in $\mathcal{D}$ and their modifications (as defined by story patterns in $\mathcal{SP}$) have to be visualized and animated using domain-specific metaphors. The only additional prerequisite, apart from the models already defined by the requirements engineer, is the definition of this domain-specific transformation as illustrated in Figure 3.13. For instance, stakeholders have to be able to interact with emails, i.e. to write and read emails or attach digital artifacts to them. By presenting the stakeholders with an icon that they are familiar with as illustrated in Figure 3.14, they are enabled to receive emails as sent by other stakeholders. This way, the stakeholder is affected by a story pattern which is executed to simulate another stakeholder. For instance, if a story pattern specifies that certain information is shared between two stakeholders via email, the recipient can receive this information via (virtual) email within the visualization as part of the simulation. Furthermore, these metaphors not only allow stakeholders to experience formal models, but also to create story patterns themselves by interacting with the virtual prototype and, thus, play in activities (cf. $\mathbf{TG}_{1B}$). Each interaction, e.g. each specific email sent to a distinct role, and each action such as *signing* an artifact is observed and saved as a story pattern which can be replayed to simulate the observed behavior.

A virtual prototype needs to present all artifacts recognizably for stakeholders who work with them on a daily basis – consequently, each instance of `Artifact` requires a picture stakeholder are familiar with assigned to it which can be shown each time the artifact needs to be represented (cf. $\mathcal{D}$ in Figure 3.3). Further, the progress of the scenario is usually marked on analog or digital versions of a specific artifact. Therefore, the representation of artifacts needs to be interactive in a way that affords stakeholders to write and draw upon them (cf. Figure 3.14b). Then, stakeholders can mark, sign, or stamp such documents recognizably for other stakeholders and, hence, hand these processed documents over to other stakeholders during the simulation.



(a) The virtual prototype enables stakeholders to interact with each other, e.g. using the concept of `Email` of $\mathcal{D}$

(b) Stakeholders are also able to work with those artifacts they are familiar with

Figure 3.14.: Our virtual prototyping approach relies on a simulation and the illustrated domain-specific animated visualization suitable for collaborative scenarios

Prototypes employed in requirements engineering usually differ quite a lot concerning the effort required for their creation [Poh10]. Consequently, whether virtual prototypes can be created quickly and inexpensively (cf. $\mathbf{TG}_{2C}$) affects their feasibility to support the iterative refinement of the collected stakeholder scenarios. Based on the formal nature of the underlying domain model and story patterns, virtual prototypes can be derived semi-automatically at virtually no cost as soon as the domain mapping and a corresponding domain-specific user interface such as the one illustrated in Figure 3.14 exist. This requires the requirements engineers to initially create the necessary visual metaphors. Generally, each concept defined in $\mathcal{D}$ that stakeholders have to interact with must have a corresponding mapping to be visualized (to notify a stakeholder of its existence) or even animated (if its modifications cannot be ignored) accordingly. The effort needed to create such metaphors cannot be estimated, since an intuitively understandable metaphor is usually not obvious, but may have to be iterated with stakeholders first. However, such metaphors can be re-used in other projects in the same or related domains as well, which makes their creation worthwhile.

Usually, stakeholders involved in a collaborative scenario neither have instantaneous access to all artifacts nor do they see every detail of what happens within the scenario. This has to be reflected in the stakeholders' visualization of the state of the scenario simulation in a realistic fashion. What has to be visualized to reproduce a distinct stakeholder's individual perspective is domain-specific. Thus, based on the stakeholders' domain, the concepts defined in the domain model $\mathcal{D}$ must be annotated to reflect which elements are visible to a stakeholder and, correspondingly, have to be presented in the stakeholder's visualization of the state a simulated scenario is in. This annotation uses the ≪visible≫ stereotype and is based on the links connecting a stakeholder's role to objects that are part of the state (illustrated in gray in Figure 3.15). Each object that is directly connected to a role via a correspondingly annotated link is considered *visible for that role*. Through prototypical iterations of these annotations for $\mathcal{D}$, the requirements engineers can quickly determine which links connecting which objects have to be represented. In the considered domain, the same annotations usually apply for all stakeholders and, hence, for all roles. As defined for $\mathcal{D}_{basic}$, an instance of `Role` requires a representation of instances of `Artifact` which are connected to this role via `has`. Further, visibility is transitive which means that, e.g. an instance of `ArtifactState` which is *visibly* connected to an instance of `Artifact` is visible for any role that `has` this artifact. Additionally, all ongoing interactions that were established before need to be included, since the interaction might be part of the rationale behind activities observed from the participant.

Based on the visibility annotations of $\mathcal{D}$, the current state s_{cur} of a simulation has to be visualized for each of the participating roles. We refer to a state s_{cur} that is reduced to what is visible for a specific role r as a *projection* $s_{cur}|_r$ or r's *perspective* of s_{cur}. For each projection for each role r, $s_{cur}|_r \subseteq s_{cur}$ has to hold. To determine what has to be represented for a specific role r, its projection $s_{cur}|_r$ has to be established for a state s_{cur}. Starting at the instance r of `Role`, all objects reachable via ≪visible≫ links

must be represented for this specific role and, thus, are elements $\in s_{cur}|_r$. This search for visible instances in s_{cur} skips over other instances of `Role`. Although they might be transitively connected via a specific `Interaction` for which r may either be the target or the initiator (connected via `to` or `from`, resp.), instances only reachable through other roles are, as in reality, not visible to r.

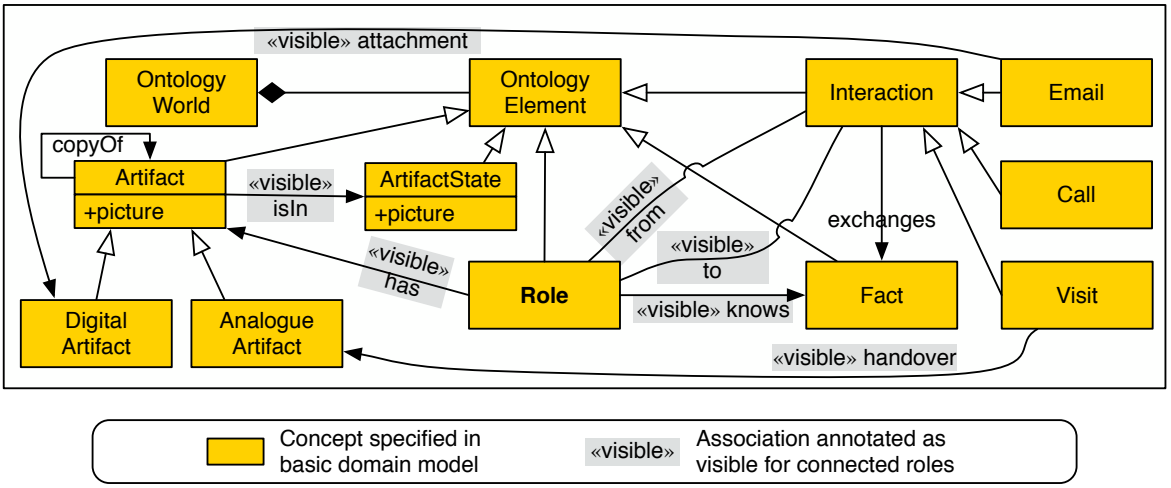


Figure 3.15.: Instances linked to a role in a state conforming to $\mathcal{D}$ are considered *visible* for that role if the requirements engineer annotated the corresponding *association* in $\mathcal{D}$ as ≪visible≫ (highlighted in gray)

To summarize, a projection $s_{cur}|_{role}$ contains the instance of *role* itself as well as all instances of `Artifact`, `Fact`, and each ongoing `Interaction` that *role* is *visibly* connected to in s_{cur} and all objects transitively visibly connected to instances as well.

3.3. Overview

The forthcoming chapters discuss the scenario simulations underlying our virtual prototyping approach. The structure is illustrated using the corresponding simulation loop in Figure 3.16. Specifically, Chapter 4 introduces the simulation loop which illustrates the approach throughout this thesis. Further, Chapter 4 describes how story patterns of different roles are replayed to enable stakeholders to experience their collaborative scenarios even in individual simulation sessions, i.e. sessions with only one participant. Afterwards, Chapter 5 extends this simulation loop by describing how story patterns are derived from observing participating stakeholders interact with each other or even simulated roles. Therefore, while Chapter 4 illustrates the *play-out* of story patterns describing activities in collaborative scenarios, Chapter 5 introduces an automatic *play-in* capability which enables stakeholders to intuitively complement their collaborative scenarios by simply enacting them. To deal with the inherent fragmentation of scenar-

ios involving multiple stakeholders and the resulting stalemates, Chapter 6 introduces a black box abstraction based on stakeholder expectations. This covers the simulation loop of our scenario simulations.

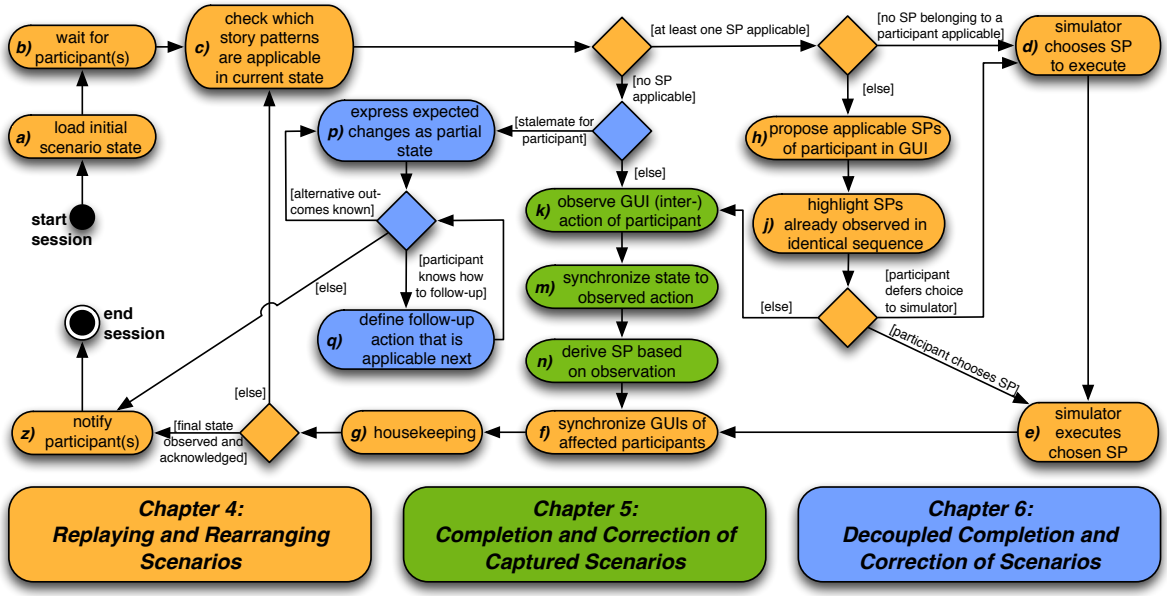


Figure 3.16.: Overview of which steps of scenario simulations are covered in which chapter of this thesis

3.4. Chapter Summary

To establish a common understanding with the stakeholders, the requirements engineers have to establish a common language. Our domain model $\mathcal{D}$, which can be gathered and iterated as simply as a glossary, covers this language gap. Since it is used as type model for situations the stakeholders can be in, the story patterns reference $\mathcal{D}$ as their type graph as well. Still, to enable stakeholders not only to talk about these situations, but to also validate the corresponding behavior they expect to occur in each situation, an intuitive representation is required.

We developed a virtual prototyping approach that, after executing story patterns specifying activities and interactions of stakeholders, visualizes how a stakeholder participating in a scenario simulation is affected relying on domain-specific metaphors. Most importantly, stakeholders can experience the content of story patterns without any of the prerequisites which are imposed by the representation of the model itself. This approach was introduced and illustrated for the domain of collaborative scenarios occurring in office environments.

4. Replaying and Rearranging Scenarios

This chapter describes the simulation concept of our virtual prototyping approach and how it can be used to validate behavioral specifications which were derived from observed stakeholder activities as they occur in collaborative scenarios. Thus, it focuses on the play-out of story patterns belonging to other roles to simulate collaborative scenarios. After identifying the roles involved in the scenarios under investigation and defining them in $\mathcal{D}$, their activities can be captured in story patterns. By being able to simulate activities of a specific stakeholder through the execution of corresponding story patterns *belonging* to him, other stakeholders may experience how they are affected by these activities. Stakeholders participating in a simulation are referred to as participants. Furthermore, each individual participant is able to point out errors concerning either whether a specific precondition, i.e. LHS, is correct before a story pattern is applicable or whether an action or interaction is correct and complete (cf. Figure 3.10). For instance, participants may point out that a necessary activity is not offered, that another activity is offered incorrectly, or that a specific artifact is not missing. Consequently, the systematic validation of story patterns representing the collaborative scenarios of the involved stakeholders affords requirements engineers the possibility to find incorrect models and identify inconsistencies within the captured behavior. Thus, while the validation ensures that all captured interactions are correct from the individual perspectives of the involved stakeholders, it also supports the verification of the overall consistency of the scenario fragments.

For such a simulation of a collaborative scenario to be successful, the already captured story patterns should suffice to cover some of the scenarios so that a participant is able to walk through a complete scenario. While this approach allows multiple participating stakeholders to interact directly during the simulation, our approach aims at quick, inexpensive single-user simulations during which all other roles are simulated as illustrated for $Role_2$ in Figure 4.1. The complex scheduling of rather infeasible group sessions as sketched by Luebbe and Weske [LW11] can be complemented through individual sessions by:

a) decoupling participants temporally from each other by simulating stakeholders who are unavailable using story patterns derived from observing those stakeholders in earlier sessions and

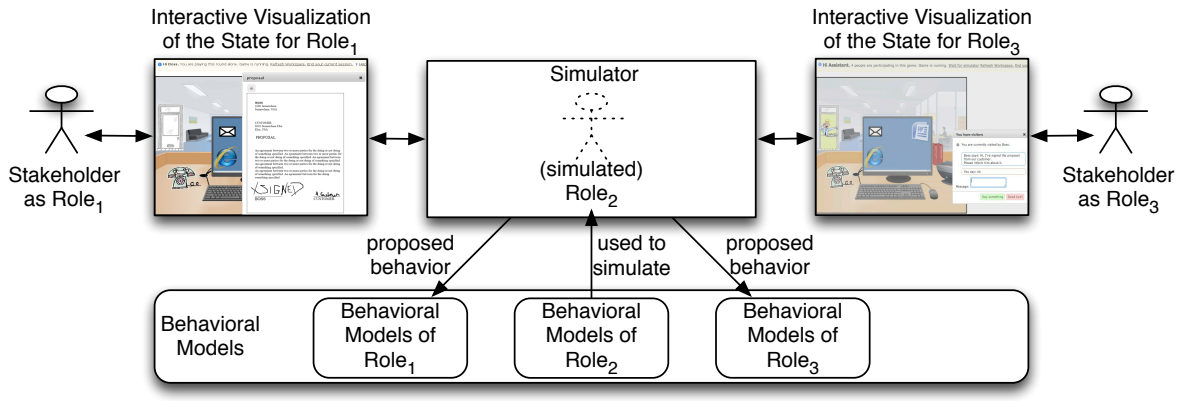


Figure 4.1.: Participating stakeholders walk through their scenarios to validate the models that represent how they interact with other roles (adapted from [GHG12b])

b) decoupling participants locally from each other by providing the interactive interface remotely which eliminates the need for the participants to be in the same room during a session

Decoupling stakeholders requires the decoupling of their interactions, as proposed by $\mathbf{TG}_1$. To achieve this goal, our approach has to simulate the behavior of individual stakeholders. Section 4.1 presents a conceptual description of our simulation approach and subsequently illustrates this approach using scenarios related to selling a movie ticket in a cinema which is based on our publication [GHG12b]. Afterwards, potential stakeholder feedback is detailed in Section 4.2. Strategies for automatically guiding the simulation are discussed in Section 4.3 which is partially based on our publication [TGRK13]. Section 4.4 concludes this chapter with a summary.

4.1. Concept

To realistically simulate actions of roles that are not played by participants, the simulation approach has to be able to manipulate the individual context of a stakeholder participating as role r. For a situation represented via the state s_i, this context encompasses everything that is visible to the role (i.e. $s_i|_r$). Only after a subsequent update of their individual visualization, can participants perceive how they are affected and react accordingly.

4.1.1. Simulation Approach

Before a simulation can be started, the requirements engineer has to define a domain model $\mathcal{D} \supseteq \mathcal{D}_{basic}$. Further, the set $\mathcal{S}$ of states needs to include at least an initial state s_{init} and a (possibly empty) set S_{term} of terminal states. Moreover, to be able to actually simulate specific roles, the non-empty set $\mathcal{SP}$ should contain story patterns belonging to the roles which have to be simulated. In the following, the individual steps necessary to guide one or more participants through a scenario are explained along the simulation loop illustrated in Figure 4.2.

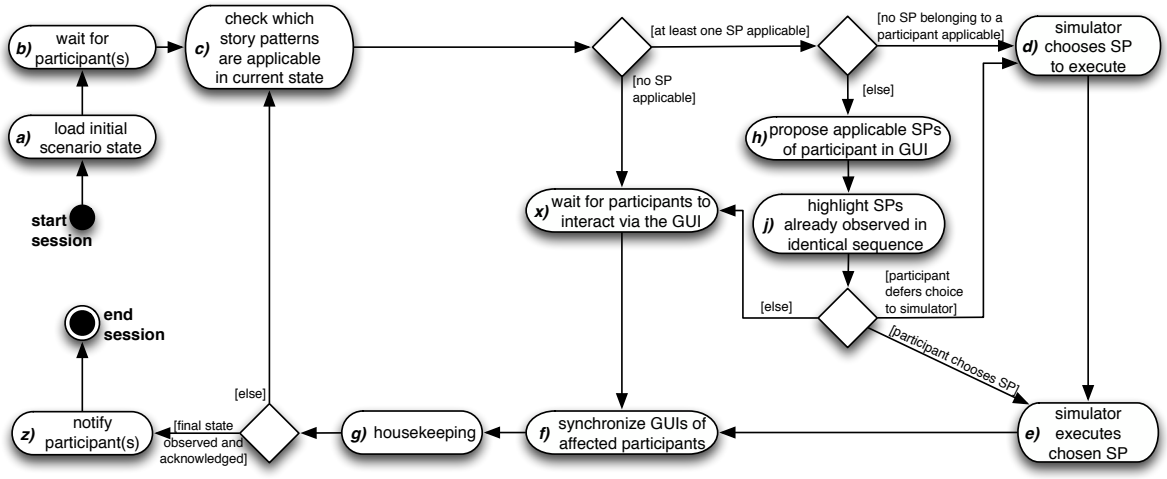


Figure 4.2.: Play-Out with the option of reliving scenarios just as they were observed before via steps *c, h, j, e* & *f* (adapted from [GHG12b])

a) Load Initial Scenario State: An initial state $s_{init} \in \mathcal{S}$ of the collaborative scenarios under investigation needs to be initialized to start a simulation. This state is a directed typed graph with the domain model $\mathcal{D}$ as its type graph and, thus, can be used to match and execute the graph transformations in $\mathcal{SP}$. This state is initialized as the *current state* (referred to as s_{cur}) of the simulation.

b) Wait for Participants: After initializing the current state of the simulation, it has to be decided which roles are enacted by participating stakeholders and which roles need to be simulated. For all roles that are not enacted, behavior observed from earlier sessions can be replayed to simulate their activities. The number of roles involved in the collaborative scenarios under investigation which were identified and defined in $\mathcal{D}$ is referred to as `n`. Thus, between `one` and `n` roles may participate in one interactive simulation session. Of course, if all roles are played by participating stakeholders, the

simulator is not required to execute behavior, but is still useful to make suggestions (cf. steps h and j in Figure 4.2) and to observe the stakeholders' interactions. If, on the other hand, no stakeholders participate, the simulation may be run without the overhead of the interactive user interface. Consequently, the number P of participants for a simulation session can be $0 \leq P \leq n$. Once all prospective participants have joined the session, the simulation can be started.

c) Check which Story Patterns are Applicable in Current State: In order to execute behavior, it must first be established which of the story patterns in $\mathcal{SP}$ are *applicable*. Usually, a stakeholder's activity has a precondition which needs to be fulfilled before the stakeholder responsible for the activity may execute it. Such preconditions are specified as objects required by the LHS of a story pattern corresponding to this activity. Only if the precondition of a story pattern is fulfilled, would the corresponding stakeholder be able to execute the activity. Of course, the same also applies for all roles which are simulated.

To know which of the activities represented in the story patterns in $\mathcal{SP}$ are applicable, the simulator needs to establish whether their respective preconditions are fulfilled in the simulation's current state s_{cur}. Since these preconditions are encoded in the LHS of story patterns, their fulfillment is checked for each story pattern by searching for a subgraph $s'_{cur} \subseteq s_{cur}$ which is isomorphic to the LHS of a story pattern $SP_i = (LHS_i, RHS_i)$ so that $s'_{cur} \cong LHS_i$. If such a match for the precondition of a story pattern is found in s_{cur}, the story pattern is considered *applicable* for the current state s_{cur} of the simulation.

This is done for all story patterns, regardless of whether or not the role they belong to is enacted by a participant. Consequently, each simulated role that has applicable story patterns associated with it would in fact be able to execute the encoded activities. Thus, at the end of step c, a set $\mathcal{SP}_{applicable} \subseteq \mathcal{SP} \cup \emptyset$ of story patterns which are applicable for s_{cur} is established along with at least one matching subgraph of s_{cur} for each applicable pattern. In other words, the simulator knows which activities of which roles can be executed next to continue the scenario. As pointed out in Section 3.1.1, different matches for a story pattern may exist when, e.g., a role has multiple identical copies of a specific artifact. If these are identical for the stakeholders, these matches are considered identical for the simulator as well and one match may be selected at random. Otherwise, additional refinements are necessary and will subsequently be pointed out by a participant if an incorrect match is chosen for execution.

h) Propose Applicable Story Patterns of Participant in GUI: If at least one of the applicable story patterns in $\mathcal{SP}_{applicable}$ belongs to one of the participants, these can be offered for execution. Only *their* story patterns are offered to the respective participants, since they are responsible for their execution and only *they* can judge whether the corresponding activity can or should be performed. Thereby, the participating stakeholders can validate whether their expectations are met. They can explicitly refute or

acknowledge whether their perception of s_{cur}, i.e. their interpretation of their individual visualization of $s_{cur}|_r$, matches the preconditions they associate with the activities offered to them.

Presenting formal story patterns to stakeholders is not promising, since their understanding of Natural Language representations of story patterns is significantly better than looking at the story pattern representation [GEHG13]. Since a story pattern is only applicable if its precondition matches, the fact that it is proposed explicitly states that the simulator considered the precondition of the activity captured in the story pattern to be fulfilled in s_{cur}. This can already be judged by the stakeholder by interpreting her visualization of s_{cur}. Therefore, it suffices to present the *changes* (elements which are either removed or added through the execution of the story pattern) in Natural Language as described in Section 2.5. For instance, if an activity represented as "*you sign the contract*" is offered to a stakeholder although she does not yet have access to a contract, she would point out that the precondition of the activity and, thus the story pattern in its current form, is invalid and has to be corrected (cf. use cases in Figure 3.10). To summarize this step, if the applicable story patterns are offered in a way that the they understand, participants may judge whether the *correct* activities are proposed based on their context.

j) Highlight Story Patterns Already Observed in Identical Sequence: After the applicable story patterns belonging to participants have been proposed, it is possible to indicate to the participants which paths through their scenarios were already covered in prior sessions. Of course, using different colors may help to differentiate between scenarios that were only partially covered once and scenarios that were covered more thoroughly and agreed upon by all stakeholders. For instance, if a sequence such as $s_{init} \xrightarrow{SP_a} s_1 \xrightarrow{SP_d} s_4 \xrightarrow{SP_e} s_5 \xrightarrow{SP_f} s_6$ has already been covered and the simulation is currently in s_5, all story patterns that are applicable and belong to the participant are proposed, e.g., SP_f, SP_g, and SP_h. However, only SP_f was observed and acknowledged in an earlier session and, thus, its NL representation that is offered in the participant's visualization can be highlighted as *covered* for this state. Consequently, the participant can explicitly choose to continue to either validate the scenario that was already observed or to explore an alternative one. The alternatives, in this case, were not observed yet, but are considered as potentially valid based on the captured story patterns. Still, if such potentially valid story patterns lead to states which are invalid from the participant's point of view, the preconditions of these story patterns need to be adjusted accordingly to exclude the subsequent invalid follow-up state.

At this point, the participant may either defer the choice of a story pattern to the simulator (step d), choose a story pattern $SP_{chosen} \in \mathcal{SP}_{applicable}$ himself (step e), or interact with the GUI and the other roles in a way that was not yet observed (step x).

d) Simulator Chooses Story Pattern to Execute: After step j, a participant may defer the choice of what to do next to the simulator. Then, it is the simulator's turn to decide which of the applicable story patterns in $\mathcal{SP}_{applicable}$ to execute. Alternatively, the same applies if none of the story patterns in $\mathcal{SP}_{applicable}$ belong to a participant, i.e. to one of the roles any participant enacts. Therefore, out of all applicable story patterns, the simulator may choose which one to execute next (referred to as SP_{chosen}). Implicitly, this choice also determines which role is simulated, i.e. the one the chosen story pattern belongs to. By default, the simulator chooses a story pattern *randomly* from $\mathcal{SP}_{applicable}$, since this mode covers all alternatives – eventually. However, to make the simulation approach more feasible, this choice can also be made more well-informed by employing metrics and specific strategies based on them (we discuss these options in Section 4.3 and in [TGRK13]).

e) Simulator Executes Chosen Story Pattern: After a story pattern SP_{chosen} was chosen either by a participant (coming from step j in Figure 4.2) or by the simulator (step d), it has to be applied on s_{cur}. Based on the application of graph transformations as discussed in Section 2.5, the application of SP_{chosen} leads to the next state s_{next} of the simulation ($s_{cur} \overset{SP_{chosen}}{\longrightarrow} s_{next}$).

x) Wait for Participants to Interact via the GUI: If no story patterns are applicable (i.e. $\mathcal{SP}_{applicable} = \emptyset$ after step c), the virtual prototype depends on inputs provided by the participants. Consequently, participants have to be able to manipulate s_{cur} to play in additional behavior via their individual interactive representation of this state.[1] Each interaction with the UI modifies the underlying state. For instance, a participant starting an interaction such as a visit with another role using the interactive visualization implicitly instantiates the corresponding concept of `Visit` as defined in $\mathcal{D}$ in the state s_{cur}, which leads to a follow-up state s_{next}. Chapter 5 discusses such GUI interactions in detail. For now, it suffices to say that a participant's interactions with the GUI can also lead to a new follow-up state s_{next} for which $s_{cur} \ncong s_{next}$ holds.

f) Synchronize GUIs of Affected Participants: After the state of the simulation has been changed either by the simulator or through GUI interactions (step e or x, resp.), participants who are *affected* by the changes have to be able to perceive these differences. Otherwise, they cannot distinguish between the former and the subsequent state, which might lead to confusion if, e.g., new story patterns are applicable (step c) and presented to them (steps h and j), although nothing changed for them.

$$\{(s_{cur}|_r) \ncong (s_{next}|_r)\} \Rightarrow \text{role } r \text{ affected} \tag{4.1}$$

[1] This may also happen if a participant interacts unexpectedly with the GUI after expected behavior was proposed (step j).

A stakeholder participating as a role r is considered *affected* by the differences between s_{cur} and s_{next} if an element within the perspective of that role changes (cf. Equation 4.1). This encompasses elements that are created, deleted or modified. If an element was created through the execution of SP_{chosen} and is *visible* to the participant enacting r, i.e. all elements $\in (RHS_{chosen} \setminus LHS_{chosen}) \cap s_{next}|_r$, its existence has to be indicated in r's representation in an intuitive way as discussed in Section 3.2. For instance, the *arrival of an email* needs to be signaled visually and audibly so that participants can experience it intuitively. However, if an element is deleted, i.e. all elements $\in (LHS_{chosen} \setminus RHS_{chosen}) \cap s_{cur}|_r$, it has to be removed from the representation of the affected participant r. Thus, it must be ensured that the domain-specific representation is capable of visualizing or animating the disappearance of these elements as part of the synchronization between s_{cur} and s_{next}. Modifications, on the other hand, are usually encoded and applied as deletion and subsequent addition of the element to be modified.

Changes within the follow-up state s_{next} of the simulation can affect any role, participating as well as simulated ones. If, for instance, a `Visit` is initiated or an `Email` is sent, not only is the initiator affected, but also the targeted role(s). For the simulated ones, however, no representation has to be provided and, hence, they do not require GUI updates. Moreover, if no role is affected (i.e. $s_{cur}|_{role} \cong s_{next}|_{role}$ holds for all roles), no updates are required.

g) Housekeeping: After each synchronization (step f), the follow-up state s_{next} reached in this round of the loop is assigned as the new current state of the simulation, i.e. $s_{cur} := s_{next}$. Then, the final states in S_{term} are tested against the new current state of the simulation to check whether the conditions for a scenario completion are fulfilled for the current simulation session. This is the case if a complete match fulfilling the condition illustrated in Equation 4.2 can be found.

$$\exists s_{term} \in S_{term} : s_{term} \cong s_{cur} \tag{4.2}$$

Since finding such a match cannot guarantee that all stakeholders are satisfied concerning the overall completion of the collaborative scenario, the participants still have to acknowledge that the matched state s_{term} is final from their distinct point of view. If one of the participants does not agree, the simulation continues to allow this participant to continue the simulation until a state can be reached, which is considered as final by all participants, i.e. they all *acknowledge* it correspondingly.

z) Notify Participants: As soon as step g confirmed that a state s_{cur} matches to a state $s_{term} \in S_{term}$ and the participants acknowledged the completion of the scenario, the simulation is considered as successfully terminated and all participants are notified within their respective visualization. As well as this type of termination, participants have to be able to directly end such a session to indicate that, from their perspective, the current state can be considered terminal. If the participants did not acknowledge a

prior match to one of the states in S_{term}, it has to be considered potentially incorrect and the state they finally agreed upon needs to be saved as a possible replacement.

To ensure that the scenario terminated correctly, the participants can then be asked whether they have any additional feedback about the scenario they just participated in.

4.1.2. Case Study: Sale of a Movie Ticket

To illustrate our simulation approach for validating behavioral models, this section discusses it using the case study of selling cinema tickets. Based on interviews with part-time movie ticket sellers and moviegoers, we specified a model $\mathcal{D}_{cinema}$ of their domain (illustrated abridged in Figure 4.3) and seven story patterns in $\mathcal{SP}$ (cf. Figure 4.4) either belonging to the role `Seller` or `MovieGoer`. These story patterns can be summarized as follows: After a moviegoer *visits* the seller (SP_v), the moviegoer needs to share the facts *time slot* (SP_a) and *movie title* (SP_b) with the seller. Knowing these facts, the seller can then create a *ticket* (SP_c) and *give* this ticket along with promotional material to the moviegoer (SP_d). Then, the moviegoer gives *money* to the seller (SP_e) and *leaves* the seller (SP_w). Furthermore, $\mathcal{S}$ consists of the two states s_{init} and s_{term} which are illustrated in Figure 4.3.

Participating as a Seller

After capturing the first session as described above, another session was observed. These two sessions as they were observed as well as three alternative transitions are illustrated as a partial state space in Figure 4.5. Now, these two sessions and the corresponding story patterns need to be validated:

I) $s_{init} \xrightarrow{SP_v} s_1 \xrightarrow{SP_a} s_2 \xrightarrow{SP_b} s_3 \xrightarrow{SP_c} s_4 \xrightarrow{SP_d} s_5 \xrightarrow{SP_e} s_6 \xrightarrow{SP_w} s_7$

II) $s_{init} \xrightarrow{SP_v} s_1 \xrightarrow{SP_b} s_8 \xrightarrow{SP_e} s_9 \xrightarrow{SP_a} s_{10} \xrightarrow{SP_c} s_{11} \xrightarrow{SP_d} s_6 \xrightarrow{SP_w} s_7$

All simulations start by loading an initial state s_{init} (step a in Figure 4.2). Afterwards, participants can remotely enter the simulation session (step b). From each state during the simulation, different scenarios may unfold depending on what the participating or simulated stakeholders decide to do.

After one complete scenario has been observed and captured, Jane participates in a validation session as a seller. All stakeholders who are not participating in this validation are simulated, thus, Jane interacts with a simulated moviegoer. The simulation starts by loading the initial state (step a in Figure 4.2) and adding Jane as participant (step b). The simulator iterates over the captured story patterns (cf. Figure 4.4) and tests whether any of them are applicable for execution (step c) on the current state s_{init} (cf. Figure 4.3) of the simulation.

All scenarios that may unfold are initiated by the moviegoer who visits the seller. Thus, no initial action is expected from the seller. The only story pattern that is

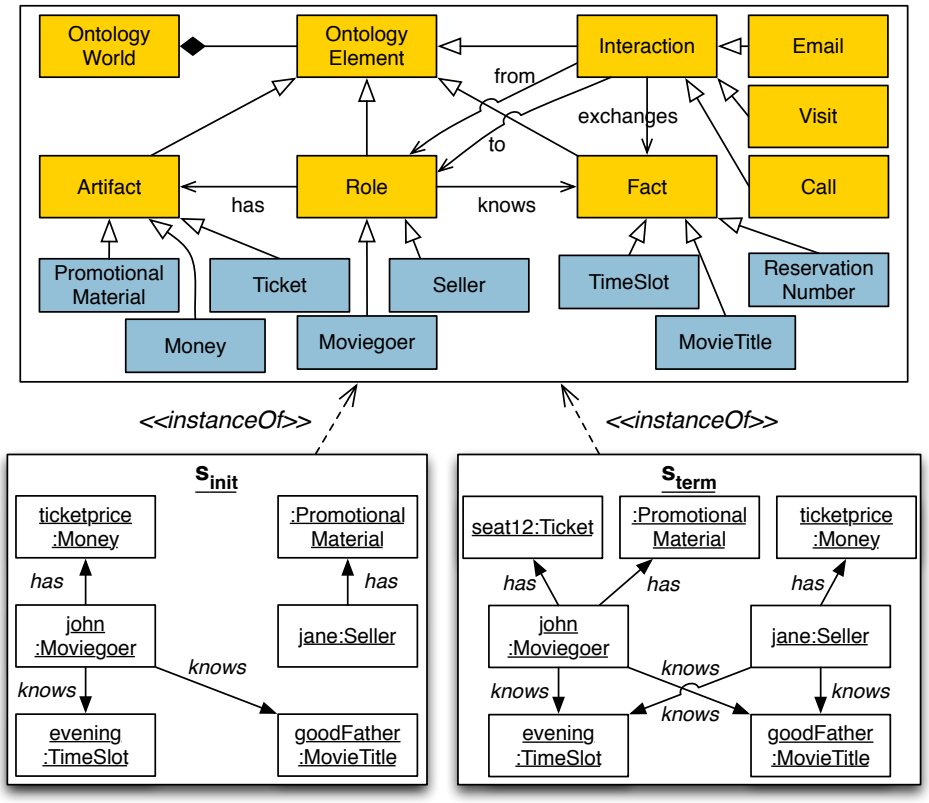


Figure 4.3.: Not only initial and terminal states of a movie ticket sale can be described using a suitable domain model $\mathcal{D}_{cinema}$ (*abridged*)

applicable to s_{init} is SP_v as discovered in step c. Since this story pattern does not belong to the participant, the simulator can directly choose it for execution (step d) and execute it (step e). This leads the simulation to the following state ($s_{init} \xrightarrow{SP_v} s_1$, cf. Figure 4.6). Then, the GUI of all participating stakeholders who are affected by the changes are updated (step f). In this case, the visit from the moviegoer has to be indicated in Jane's GUI. Since no final state can be matched (step g), the simulation continues by checking which story patterns are applicable in s_1 (step c).

Only after a visit has been established, is the simulated moviegoer able to share the information that the seller requires for the creation of the corresponding ticket: either the moviegoer tells Jane which movie he wants to see or in which time slot he wants to see it (SP_b or SP_a in Figure 4.4, respectively). The simulator may execute either one of these models to simulate the behavior expected from the moviegoer.

After the simulator randomly chooses SP_a (step d) and executes it (step e), the simulator is in the follow-up state ($s_1 \xrightarrow{SP_a} s_2$). In s_2 (Figure 4.6), the seller has access to the fact of the moviegoer's chosen time. This has to be indicated for Jane in her GUI as part of the GUI synchronization for all affected participants (step f). Since this

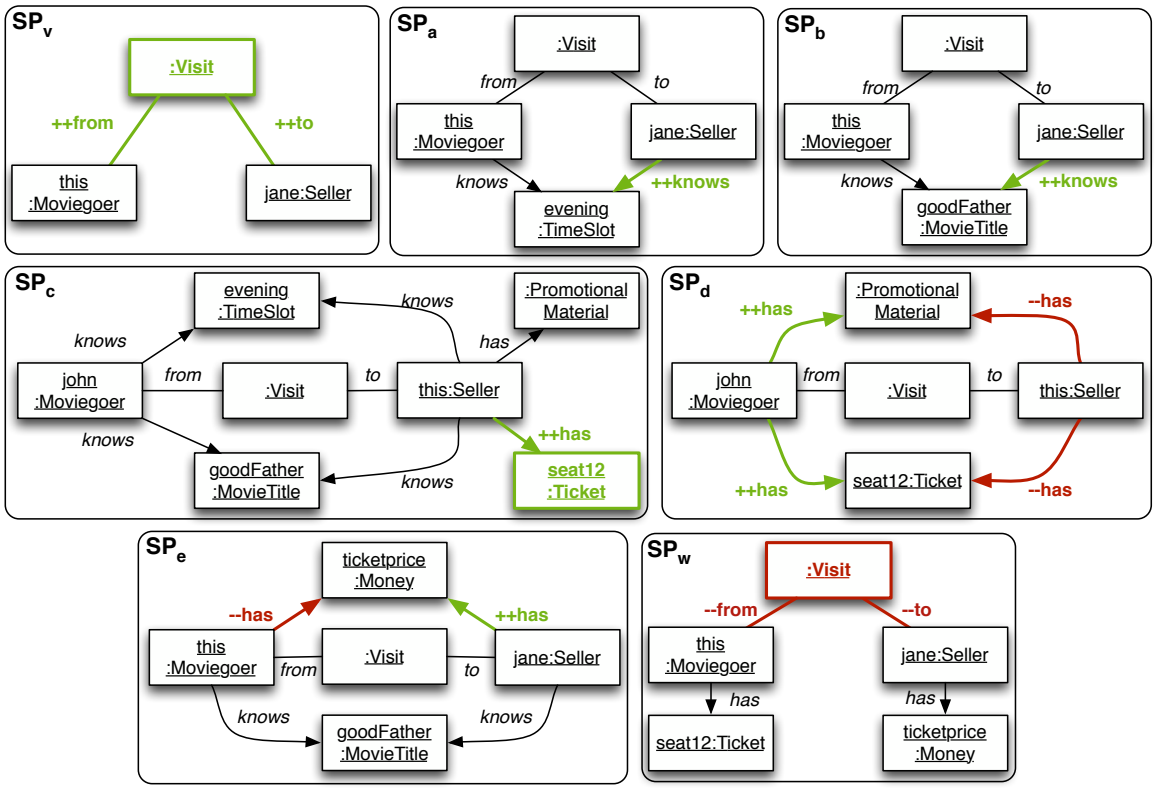


Figure 4.4.: Observing a ticket sale and deriving different pieces of the puzzle

information does not suffice for the ticket purchase to take place, the moviegoer has to share which movie he wants to see as well. Consequently, the simulator (not having reached a final state yet) checks which story patterns are applicable in the current state s_2 (step c). The only other activity (SP_b – "*the moviegoer tells the seller the title of the movie*") is chosen and executed, leading to the state s_3 and another update in Jane's GUI ($s_2 \xrightarrow{SP_b} s_3$).

As pointed out before, participating stakeholders can acknowledge or refute behavior of other stakeholders if it directly affects them. Thus, if the moviegoer were able to create or access a ticket without any interaction with Jane, she could not comment on it, since she would not be affected in this scenario. If, however, he takes the ticket she offers him without paying, she would most certainly object, thereby identifying a conflict.

Only after both facts were shared on behalf of the simulated moviegoer (s_3 in Figure 4.5), is Jane able to create a ticket (SP_c in Figure 4.4) for the moviegoer. Thus, after checking which behavioral specifications are applicable (step c), only SP_c and SP_e fit. Since SP_e belongs to the role moviegoer, it is not proposed to Jane. SP_c, on the other hand, belongs to the role seller and, therefore, is proposed to Jane in her visualization (step h). Other story patterns belonging to seller, which might be applicable, would also

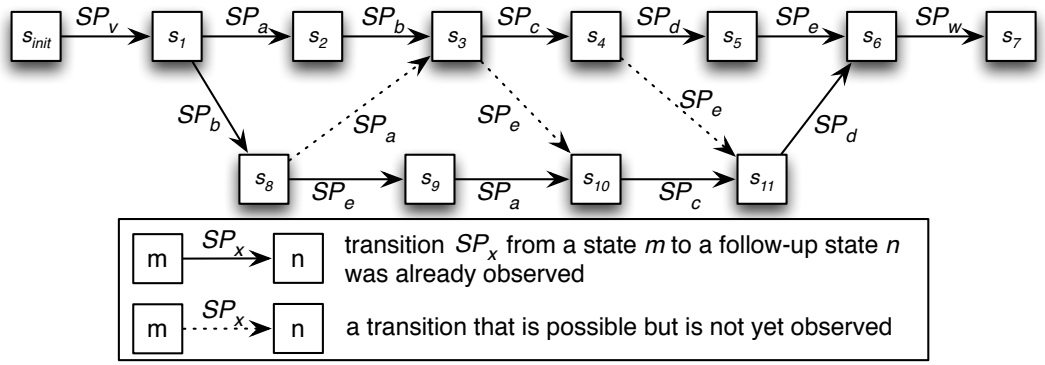


Figure 4.5.: After validating two sessions, three alternative transitions are still possible

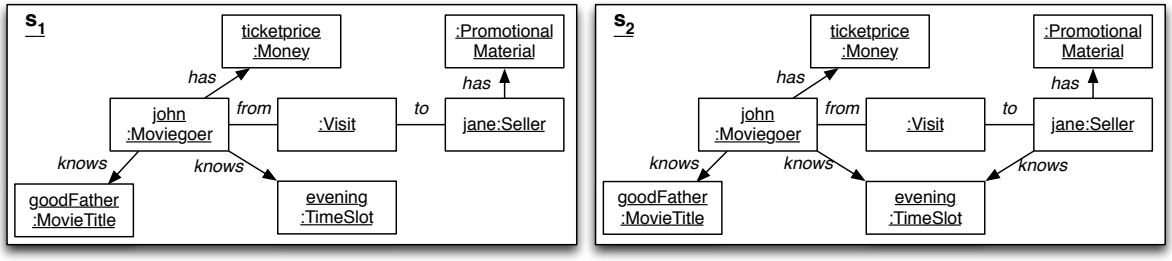


Figure 4.6.: State specifications *before* (s_1) and *after* (s_2) John tells Jane when he wants to see a movie (also reachable by applying SP_a from Figure 4.4 on s_1)

be proposed to Jane, representing all alternative scenarios which were already observed and are still possible. If any of the applicable story patterns were already observed in exactly the same context, i.e. after exactly the same sequence of story patterns, they would be highlighted (step j) in Jane's visualization.

At this point, Jane can choose to interact with her representation $s_{cur}|_{seller}$ of the current state to create the ticket manually (step x). It is also possible to tell the simulator to simply replay the already captured pattern SP_c (step e). If multiple story patterns are proposed to her, she may instead defer the decision of which pattern to execute to the simulator (step d). For this round, Jane continues by choosing SP_c for execution, thereby creating a ticket ($s_3 \xrightarrow{SP_c} s_4$). After the synchronization between her GUI and the current state of the simulation (step f), she can see the ticket and recognizes that she can hand it over to the moviegoer.

In state s_4 (cf. Figure 4.5), SP_d and SP_e are identified (step c) as applicable. Again, only story patterns belonging to the seller are proposed to Jane (step h) – in this case SP_d which she chooses to hand over the ticket and the promotional material to the moviegoer (step e). Since the GUIs of all affected participants are updated (step f), Jane consequently looses access to both artifacts.

As mentioned before, the story pattern representing the payment of a ticket (SP_e in Figure 4.4) belongs to the simulated moviegoer and, therefore, is not considered as long as a participant may enact behavior. Consequently, while it was applicable in s_3 and s_4, it was up to the participant to decide how to continue. Only if the participant had deferred the choice to the simulator might SP_e have been chosen and executed. Now, however, it is the only applicable story pattern in the simulation's current state s_5 (cf. Figure 4.5). The simulated moviegoer is now able to pay (steps *c, d,* and *e*) – much to the relief of Jane who receives and perceives the money in her GUI (step *f*). In the subsequent step *g*, the simulator cannot match the final state s_{term} (cf. Figure 4.3) since the visit is not terminated yet. Consequently, step *c* establishes that the simulated moviegoer can terminate the visit using SP_w, the only applicable story pattern. After the simulator chooses and executes SP_w (steps *d* and *e*), and Jane's GUI is updated (step *f*). At this point, s_{cur} is a complete match for s_{term} (step *g*) and Jane acknowledges the successful termination of the scenario.

To summarize, Jane experienced the behavior of a simulated moviegoer while participating as a seller. All handovers were causally correct and as expected. Moreover, she not only acknowledged the correctness of the moviegoer's story patterns, but also that her story patterns were correct and the overall scenario, which she experienced, was consistent.

Pointing Out and Correcting Factual Errors

Since most of the existing pieces of the overall scenario puzzle have already been captured, all involved stakeholders can validate whether and how they fit together (i.e. which sequences in $\mathcal{P}(\mathcal{SP})$ are valid). By acknowledging or refuting the correctness of sequences of these story patterns, all invalid sequences can be explicitly excluded by slightly modifying the preconditions of these patterns or removing them completely. For instance, it might be against the cinema policy to hand over any tickets before they were paid for by the customer. After another stakeholder in the role of a seller joins in a session with a participant as moviegoer, this second seller points out that ticket may only be handed over *after* the moviegoer has paid for it. Consequently, to enforce such a policy, the preconditions of when to hand over the ticket need to be adapted. In other words, the LHS of SP_d simply needs to exclude all states in which the seller did not yet receive the money – a trivial addition to this LHS as presented in Figure 4.7. After the revision of SP_d, the adapted pattern SP'_d can only be matched within states in which the moviegoer paid first before receiving the ticket. The resulting partial state space illustrating the valid story pattern execution sequences are presented in Figure 4.8. By replacing SP_d with SP'_d, the first session becomes incomplete, since the state s_5 cannot be reached anymore. Furthermore, the transition $s_{11} \xrightarrow{SP'_d} s_6$ is available, but has not been observed and validated yet. Still, since the LHS has become stricter and,

thus, is applicable in fewer states than before, it can still be considered as valid for the remaining states in which it is applicable.

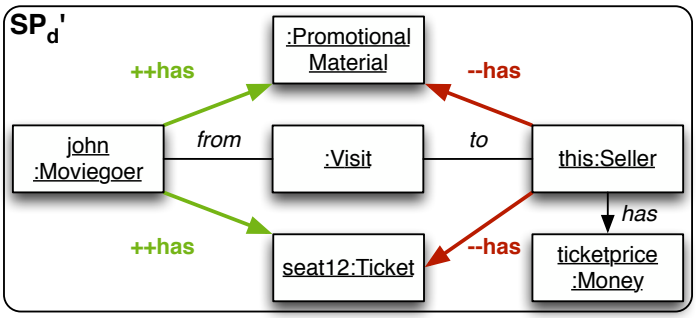


Figure 4.7.: In this revision of SP_d (Figure 4.4) the seller must get the money first

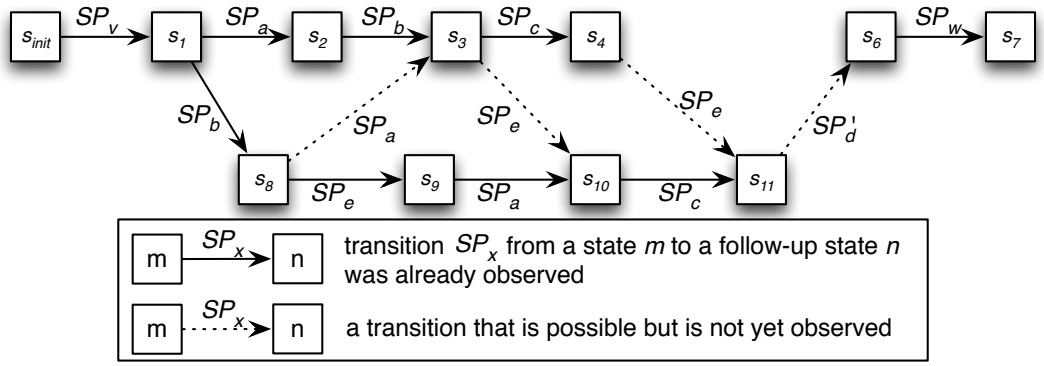
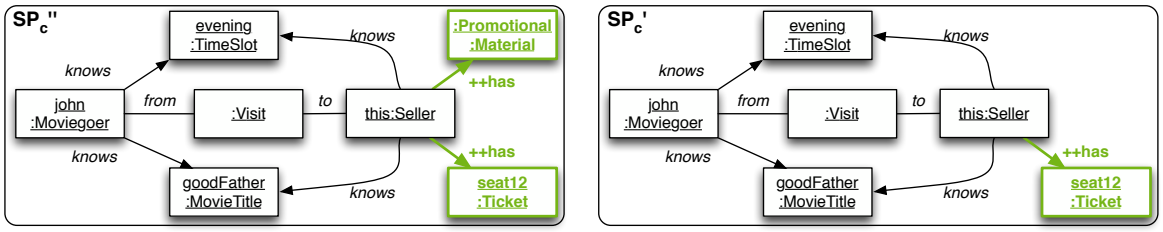


Figure 4.8.: By adapting SP_d to SP'_d, transition $s_4 \xrightarrow{SP_d} s_5$ is no longer available and $s_{11} \xrightarrow{SP'_d} s_6$ has to be validated again since it was not observed in this sequence

Generally, the fact that a story pattern SP_a might be applied on a state s_i ($s_i \xrightarrow{SP_a} s_j$) implies that a match for the preconditions of SP_a must exist ($\exists s'_i \subseteq s_i : s'_i \cong LHS_a$). If either the resulting state s_j or the application of SP_a leading to it is considered incorrect, this application has to be inhibited. By expressing additional preconditions in the story pattern's LHS, the requirements engineer can modify SP_a in such a way that $SP'_a = (LHS'_a, RHS'_a)$ is no longer applicable in s_i ($\nexists s'_i \subseteq s_i : s'_i \cong LHS'_a$). Elements which are added or removed from a story pattern's LHS have to be included or removed from the RHS as well to keep the activity that is encoded in the story pattern consistent: $(RHS \setminus LHS) \cong (RHS' \setminus LHS')$ has to hold. Otherwise, if RHS is not adapted, all elements added only to LHS are explicitly deleted if they are not in RHS as well.

On the other hand, a requirements engineer might need to explicitly include a state s_j which does not yet contain a matching subgraph for a story pattern $SP_b = (LHS_b, RHS_b)$, i.e. $\nexists s'_j \subseteq s_j : s'_j \cong LHS_b$. To ensure that such a subgraph s'_j can be found, all elements



(a) Adapting only LHS leads to new side effects (b) Adapting RHS as well conserves the activity

Figure 4.9.: To remove promotional material from SP_c's precondition, it has to be removed from the RHS as well – otherwise, the activity is changed

in LHS_b which cannot be found in s_j have to be removed from LHS_b. This, however, is only possible as long as these elements are not modified as part of the story pattern's activity, i.e. the elements which may be removed must be in $(RHS \cap LHS)$. If this is ensured, the resulting *less restrictive* precondition which is applicable in the same states as before plus *at least* s_j is the following: $LHS'_b := LHS_b \cap s_j$. Only by removing the same elements $(LHS_b \setminus s_j)$ from the RHS as well, can the enacted changes be kept consistent: $RHS'_b := RHS_b \setminus (LHS_b \setminus s_j)$. For instance, if a participant points out that the creation of a ticket (SP_c in Figure 4.4) should also be possible in situations in which the seller does not have promotional material ($\in LHS_c \cap RHS_c$), removing it from LHS_c alone would lead to an incorrect story pattern which creates a ticket *and promotional material* the seller has access to (illustrated in Figure 4.9a). By removing the promotional material from RHS_c as well, the original activity represented in the story pattern remains intact (cf. Figure 4.9b) and would be applicable in the desired state(s).

4.2. Stakeholder Feedback in Validation Sessions

For their synthesis of protocol-based models, Desai et al. [DMCS05] emphasize the importance of what they refer to as *local policies* for the enactment of interactions. They define a role's local policy as its (usually private) individual business logic which controls the role's participation and, thus, its decisions in the different scenarios as part of a process. The same goes for interactions between individual stakeholders. Due to such a private decision rationale, interactions with other stakeholders might seem similar to interacting with a *black box*, i.e. being only able to observe actions and reactions on inputs provided beforehand. Consequently, in the validation of story patterns and the scenarios they cover, we have to distinguish between different kinds of observations.

4.2.1. Story Patterns Belonging to a Participant's Role

A story pattern SP_i that is proposed correctly is a story pattern which represents an activity that is applicable form the point of view of the participant whose role SP_i belongs to. Thus, judging from the RHS of each story pattern that is proposed, the participant can recognize and point out any incorrect preconditions (cf. use cases in Figure 3.10). If, for instance, the simulator arrived in a state s_2 ($s_{init} \xrightarrow{SP_u} s_1 \xrightarrow{SP_a} s_2$, cf. Figure 4.5) and a stakeholder participating as a seller was only informed about the time slot, she would recognize and point out that an incorrectly proposed SP_c (*"You (Seller) create seat12(Ticket)"*, cf. Figure 4.4) should not yet be available. In s_3, on the other hand, the seller received both required facts and the stakeholder would acknowledge this proposed activity as correct for this situation. Thus, by acknowledging or refuting the correctness of sequences of these story pattern as well as when a story pattern is proposed (or when it is not), all invalid sequences can be explicitly excluded by slightly modifying the preconditions of these patterns as demonstrated in Section 4.1.2 or removing them completely. The more complex the preconditions in a story pattern's LHS, the less often it is applicable, observed, or executed during validation sessions.

4.2.2. Story Patterns Affecting a Participant's Role

Some of the story patterns which do not belong to a participant can still affect the role in which the stakeholder participates. If a participating stakeholder interacts several times with a simulated role, the participant cannot know how many different activities of how many roles have to be simulated in between the interactions which she is involved in. The participant can only perceive how she is affected by these activities if the representation $s_{cur}|_r$ of her role is updated accordingly. Consequently, she may only point out whether a state s_i in which the simulation arrives is valid based on her perception of this state and her experience. Generally, after a participant interacts with a simulated role r_{simu}, she usually has to wait for a reaction which arrives in a state s_i in which she is affected before she can continue. Otherwise, the participant's representation of a sequence $s_{cur} \xrightarrow{SP_1} s_a \xrightarrow{SP_2} s_b \ldots\ s_n \xrightarrow{SP_m} s_i$ is mostly identical ($s_{cur}|_r \cong s_a|_r \cong s_b|_r \ldots \cong s_n|_r$) until the state s_i is reached for which $s_n|_r \ncong s_i|_r$ holds. If this is the case, all story patterns in between cannot be judged directly, since the participant may not even know how many story patterns were executed to simulate the parts of the scenario she is *not* involved in. For instance, a participating seller arrives in s_5 in which she already handed over the ticket but was not paid yet ($s_{init} \xrightarrow{SP_u} s_1 \xrightarrow{SP_a} s_2 \xrightarrow{SP_b} s_3 \xrightarrow{SP_c} s_4 \xrightarrow{SP_d} s_5$, cf. Figure 4.5). Then, a simulated moviegoer might execute a story pattern that allows him to leave without paying ($s_5 \xrightarrow{SP'_w} s_i$ using a correspondingly modified version of SP_w). Since the moviegoer's visit ended, this story pattern affects the participant ($s_5|_r \ncong s_i|_r$). While she does not know what else might have happened in between, she is still able to refute the observed scenario by pointing out that the state s_i is invalid. Therefore,

the sequence of activities of other roles in between the last state s_5 right after her last activity (after $s_4 \xrightarrow{SP_d} s_5$) and the last state s_i in which she was affected in an incorrect manner contains at least one incorrect story pattern.

To summarize, although identifying errors in sequences of story patterns which are executed non-transparently is possible, a participant must be affected to pinpoint the error. Even than, identifying which specific story pattern is incorrect is not easy.

4.2.3. General Feedback

Generally, the types of errors which participants may point out during the simulation include, but are not limited to, activities which are incorrectly represented by a story pattern, incorrect preconditions which lead to a story pattern being applicable in too many or to few situations, or inconsistencies between different stakeholders. Through multiple iterations with different participants, the requirements engineers can validate which behavior (represented by a corresponding story pattern $SP \in \mathcal{SP}$) is considered as suitable in which situations and, thus, should be applicable in which states. By adjusting the preconditions of story patterns which participants identified as incorrect for specific states, it can be ensured that they are only applicable in valid states the stakeholders usually find themselves in during the scenario. For instance, if a participant argues that a specific story pattern, whose natural language representation is offered for execution, cannot be executed yet on account of information being missing, the precondition of this story pattern has not been restrictive enough. Consequently, the corresponding information has to be introduced in the precondition so that the story pattern is only applicable if the restriction pointed out by the participant is fulfilled.[2] Apart from the errors participants point out explicitly, the remainder of what is shown to them is considered acknowledged. Thus, if all participants play through a scenario without complaints, it is considered agreed upon by these stakeholders. Further, our approach allows stakeholders to point out activities which are proposed although they should not be applicable (incorrect options). Additionally, stakeholders may also be able to enumerate which activities they would expect to be possible and, thus, implicitly recognize missing story patterns. Consequently, the simulation allows stakeholders to recognize errors of *commission* and *omission*, respectively [BP84].

Moreover, the final feedback of participants after a scenario is terminated can indicate how satisfied they are concerning the coverage so far. Such feedback is gathered using freeform text with a suitable prompt.

[2] If the required concepts have not yet been captured, $\mathcal{D}$ is extended correspondingly (Figure 3.8).

4.3. Strategy-Driven Exploration of Stakeholder Scenarios

So far, the concepts presented in this chapter have only discussed how to randomly choose one of the applicable story patterns for execution in step *d*. Based on the generation of a partial *state space* [Val91] (also referred to as *reachability analysis* [LCL87]), this section discusses strategies for deciding which story pattern to choose and how to continue.

4.3.1. Strategies Based on a Limited Look Ahead

The simulation of collaborative scenarios can be guided more effectively if the consequences of the applicable story patterns are known. For instance, while deadlocks should be avoided during the elicitation of new behavior, it may be useful to guide stakeholders into such a deadlock during a validation session to allow them to pinpoint the erroneous behavior or assumptions leading to it. For our purposes, the state space consists of all reachable states and the transitions between them starting from s_{init} and relying on the story patterns in $\mathcal{SP}$. In other words, the state space we want to explore represents the possible scenarios which may unfold based on s_{init} and $\mathcal{SP}$. If the *complete* state space is explored, all eventually occurring consequences and their side-effects are known. However, it is not always reasonable to compute all possibilities beforehand, especially since the state space might be too big to handle or even infinite. Additionally, the state space supports a guided simulation only in cases in which the participant's actions were known in the form of expected inputs for the simulation. Consequently, the state space might be outdated if a state that was not reachable before is entered through a participant's actions in step x (Figure 4.2). In a survey of simulation tools for requirements engineering [SRB+00], Schmid et al. distinguish between *random* and *limited* depth exploration. For the remainder of this thesis, we refer to the result of a limited depth exploration as a *look ahead*. As described in [Ric11, Teu11], a look ahead which starts at the current state s_{cur} and with a depth of `n` contains all states which can be reached from s_{cur} within the execution of sequences of up to `n` story patterns. Therefore, each time the simulator checks which of the story patterns in $\mathcal{SP}$ are applicable for a state s_{cur} (step *c*), the simulation implicitly calculates a look ahead with a depth of `1`.

As we pointed out in [TGRK13], two distinct kinds of strategies cover the choice of which story pattern to execute next: strategies which rely on the *structure of the look ahead* and strategies which base this decision on specific *properties of individual states*.

Structure of the Look Ahead: Based on the structure of a look ahead generated from s_{cur}, it can be seen how many different diverging paths unfold after each subsequent transition. Depending on the depth of the look ahead, these paths may even be converging later on, until one of the terminal states of the collaborative scenario is reached.

For each reachable state s_j in the look ahead, the following considerations apply. If there are no *outgoing* transitions, i.e. no story patterns are applicable, this state might be invalid, especially if it was already visited in a former session. However, if all *incoming* transitions are purely speculative and, thus, have neither been *observed* nor *excluded* in a prior session, then it is important to process this state. If a suitable participant accepts the transition as valid, it is considered observed and, thus, correct. The exclusion of such a state through the adaption of the LHS of the previously executed story pattern, on the other hand, was already covered in Section 4.1.2.

Moreover, a look ahead can be used to compute which sequences offer the most alternative outcomes – while one sequence might lead to a diverse total of ten alternative states at the n^{th} step, another sequence might be rather straight-forward resulting in only *one* state. As Richter [Ric11] points out, if a specific state s_j has many outgoing transitions, the preconditions of the corresponding story patterns might be inaccurate or not restrictive enough. By choosing the story patterns corresponding to the sequence leading to s_j as identified in the look ahead, a participant may be steered towards s_j. Once arrived in this state, specific participants may judge whether the applicable story patterns are suitable based on their experience concerning the corresponding activities. In summary, strategies which base their decision of how to continue purely on the structure of the look ahead are suitable for the exploration of collaborative scenarios if they always choose diverging sequences. To validate behavior leading to alleged deadlocks, however, a strategy always choosing sequences which converge in the look ahead may be employed.

Scoring Individual States within the Look Ahead: After a look ahead has been generated, different scoring algorithms can be applied to compare the reachable states within the look ahead. Instead of only deciding based on the structure, individual objects in distinct states can be investigated and compared. Hence, although such computations introduce overhead, values of specific attributes can be summed up to evaluate whether a state s_i is closer to a previously specified goal than another state s_j [Teu11].

For instance, an *attribute-based* form of such a strategy has to score a state based on the value of one or more specific attributes such as a role's *budget*. Depending on how high the value of the corresponding attribute for budget becomes, the strategy would score such a state higher (or lower, depending on which situation the stakeholder should experience). By introducing individual weights $w_i \in \mathbb{Z}$ to the attributes a_i, in which the requirements engineer is interested, more complex scoring functions can be used. A simple example is illustrated in Equation 4.3, which can be used to sum up all resources times their individual weight in terms of how these values changed between the simulation's current state s_{cur} and a follow-up state s_x in the look ahead (adapted from [TGRK13]). While a_i^x represents the value of attribute a_i in state s_x, a_i^{cur} corresponds to this attribute's value in s_{cur}. After all states within the look ahead have been scored correspondingly, the state with the highest score can be chosen. This, in turn, also

chooses a sequence of story patterns which, when applied in the same order, arrive at this state.

$$score(s_{cur} \rightarrow s_x) = \sum_{i=1}^{|Attributes|} w_i \times (a_i^x - a_i^{cur}) \tag{4.3}$$

Alternatively, a binary *object-based* form of such a strategy might score states, in which a specific desired concept such as a *ticket* exists higher than states which do not contain a ticket. Consequently, as soon as a state containing a ticket is identified within the look ahead, such a strategy would score this state higher and try to reach it by deciding for story patterns belonging to the shortest sequence leading to this state.

The peculiarities of individual states within a look ahead are ignored by strategies which only focus on the structure of the look ahead. Strategies which score these states, on the other hand, allow the requirements engineers to define specific goals or characterize situations (e.g. focus on states in which the moviegoer has a ticket) which the simulator then tries to reach.

4.3.2. Reducing the Participants' Downtime

Imagine a situation s_{cur}, in which story patterns are applicable, but none of them belong to a participant. Thus, the simulator has to decide which story pattern to execute (step *d*). As long as the context of this participant remains unchanged, none of his story patterns will be applicable. However, the more story patterns are executed before a participant is eventually affected in a follow-up state s_i (i.e. $s_{cur}|_r \ncong s_i|_r$), the more uncertainty is introduced in between s_{cur} and s_i. Specifically, as pointed out in Section 4.2.2, while the participant may recognize that an incorrect state is reached, he is not able to pinpoint the error to one of the story patterns in between. Consequently, by ensuring that a participant will be affected by the smallest number of story pattern executions in between, this uncertainty can be reduced. If an incorrect state is identified, fewer story patterns should be considered suspicious. Furthermore, executing fewer story patterns decreases the probability of an erroneous sequence. Depending on the time it takes to find, choose, and apply the story patterns in between, the simulation remains responsive instead of paused from the participant's perspective.

Whether or not a participant's role is affected can be deduced from the changes encoded in each applicable story pattern. Any of these story patterns affect a participant by either creating, modifying, or deleting an association which has the participant as target or source. Alternatively, a participant may also be affected by the creation, deletion, or modification of objects directly connected to the corresponding role. Such a check can either be performed ad hoc for all the story patterns that are applicable for the current state s_{cur} of the simulation, or by finding the shortest sequence of story patterns within a limited look ahead.

4.4. Chapter Summary

In this chapter, we presented our approach of replaying story patterns to simulate the interactions of different stakeholders in collaborative scenarios. Through such a simulation, participating stakeholders can acknowledge or refute sequences of story patterns which represent activities as they were (potentially) observed in previous stakeholder sessions. Thereby, the simulator decouples stakeholder interactions. The validation of such interactions is conducted using an animated simulation suitable for all involved stakeholders. At the same time, the simulator can use the story patterns to propose speculative sequences and, thus, to explore alternative scenarios. Further, by defining and selecting appropriate strategies, requirements engineers can guide stakeholders towards specific situations which were not validated yet or marked as being in conflict. In combination with a short feedback loop, this strategy-driven simulation approach supports requirements engineers to elicit, understand, and validate such collaborative scenarios.

Nevertheless, a non-empty set of story patterns was among the preconditions of this chapter. The following chapter discusses how our approach overcomes this precondition and, thereby, becomes more feasible through the automatic generation of story patterns based on observations.

5. Completion and Correction of Captured Scenarios

Apart from automatically *replaying* story patterns to allow stakeholders to validate the models in $\mathcal{SP}$ as described in Chapter 4, our approach also facilitates the semi-automatic *elicitation* of such models. By steering the simulation toward incomplete scenarios, i.e. sequences of story patterns which do not lead to a successful termination of the scenario, our approach allows participants to *play in* new behavior and complement each others' scenarios. While the simulator replays story patterns in a way that enables it to ask a participant *"Is (this activity in) this state correct?"*, the simulator can also provide the context to ask a participant: *"From this current situation, how do you continue?"* and *"What are your next steps toward the successful termination of the scenario?"* In such a situation, a participant can enact the subsequently required activities within her individual intuitive representation. Hence, the stakeholders are empowered to complement their scenarios step by step in a decoupled manner. Further, by employing simulation strategies, participants can be steered into scenarios which have not yet been completed or specific situations which requirements engineers are interested in (cf. Section 4.3.1). Consequently, the overall coverage of how the stakeholders interact increases since more fragments of these collaborative scenarios can be observed and elicited.

By interacting with each other or simulated roles using the intuitive representation, participants may play in new formal models without being directly confronted with them. Even if an elicitation starts without any behavioral models which the simulator may use to simulate other roles, complete scenarios can be played through if all involved roles are played by participating stakeholders. Thus, the elicitation of many scenarios is possible even without the simulator – solely by relying on the derivation of formal models based on observing the participants' interactions. The only prerequisite for such a scenario simulation is a state in which the simulation of the scenario can be initialized.

While Chapter 4 introduced the simulation concepts necessary to enable stakeholders to validate scenarios, this chapter discusses how stakeholders may complete scenarios and correct erroneous story patterns. In this chapter, Section 5.1 which is partially based on [GHG12b] explains our concept of allowing stakeholders involved in highly collaborative scenarios to intuitively play in story patterns and, thus, add their individual behavior to $\mathcal{SP}$. Further, this section also illustrates the concepts building on the cinema case study introduced in Section 4.1.2. While Section 5.2 discusses the restrictions on the story patterns derived from such an elicitation session, the merging of the resulting story patterns is presented in Section 5.3. Section 5.4 summarizes this chapter.

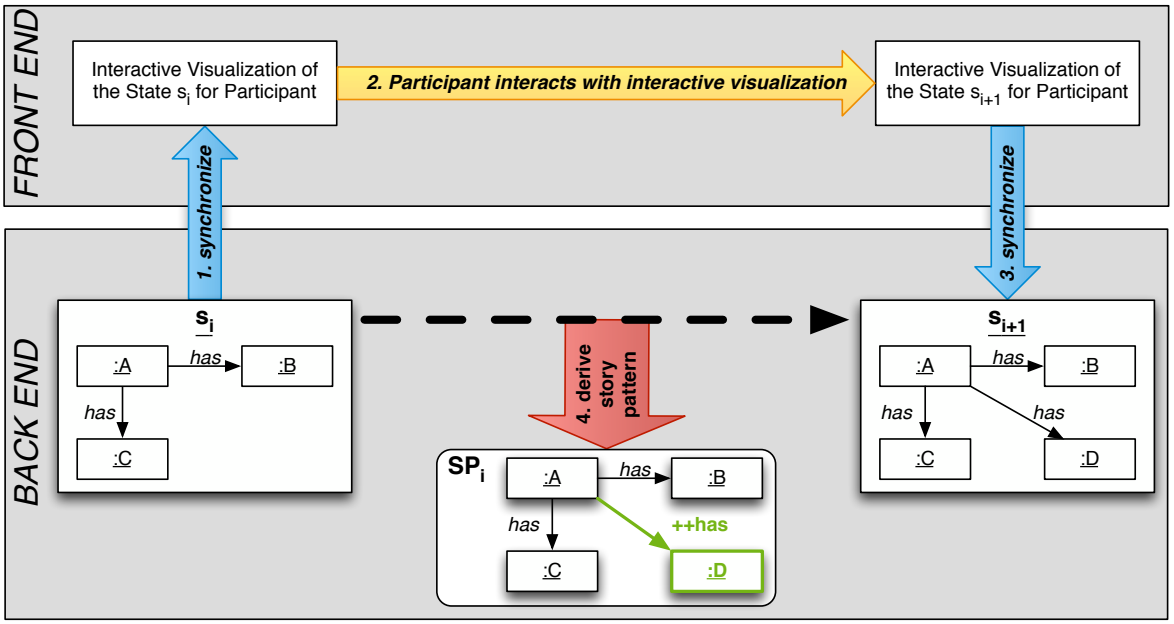


Figure 5.1.: When a state s_i is visualized (*1*), participants can interact with their visualization (*2*), thereby manipulating the underlying state (*3*) so that a story pattern SP_i can be derived (*4*) based on the differences between s_i and s_{i+1}

5.1. Concept

By providing intuitive interaction elements as part of the stakeholder specific state visualization (*1.* in Figure 5.1), participants can (indirectly) manipulate the underlying state s_i of the simulation by interacting with other roles or even documents (*2.* in Figure 5.1) to arrive at the follow-up state s_{i+1} (*3.* in Figure 5.1). The difference between this pair of states (s_i, s_{i+1}) represents the intended action of the participant. Consequently, this action can be formally captured in a story pattern SP_i (*4.* Figure 5.1) which, if applied to the state s_i, leads to s_{i+1} as observed in the simulation session. Thus, the actions observed during a simulation are captured formally in a way that allows the simulator to execute them in later sessions to replay what the participant was observed doing.

5.1.1. Simulation Loop Including Stakeholder Input

To include an observation-based elicitation into the scenario simulation concept, the simulation loop has to be extended correspondingly. Thus, the step *x) Wait for Participants to Interact via the GUI* shown in Figure 4.2 is refined by three additional steps *k*, *m*, and *n* as illustrated in Figure 5.2. The remainder of the simulation loop remains the same.

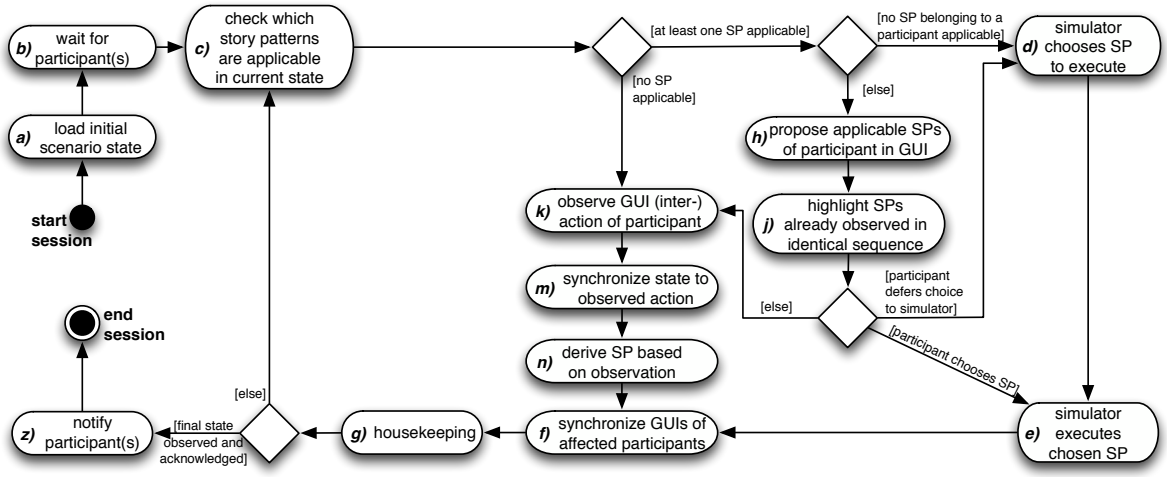


Figure 5.2.: After refining step x into k, m, and n, the simulation loop supports not only the replay of stakeholder activities but also the *play-in* of new story patterns

k) Observe GUI (Inter-)Actions of Participant: By observing how a stakeholder participating as role r interacts with her representation $s_{cur}|_r$ of the current state s_{cur} and mapping these interactions back into the state itself, our approach empowers stakeholders to manipulate the underlying state of the simulation in an intuitive way. Thus, the participant's interaction with her representation during the simulation leads to events which have to be interpreted as updates for the underlying current state. Apart from events created within the representation, requirements engineers can also directly observe the participants' interactions, while the participants enact their collaborative scenarios.

m) Synchronize State to Observed Action: Based on the domain mapping between the affordances offered in the intuitive representation and changes that the participants can execute on the current state s_{cur}, the simulation needs to progress correspondingly to a follow-up state s_{next}. For instance, if a participant interacted with another role via email, the corresponding domain concept `Email` is instantiated accordingly as part of the modification leading to s_{next} – based on the mapping between domain-specific GUI and the domain-specific model.

n) Derive Story Pattern Based on Observation: Based on the differences between s_{cur} and the subsequent s_{next}, the observed activity can simply be extracted and captured in a *complete* story pattern, i.e. $SP_{derivedAll} = (s_{cur}, s_{next})$ based on both states. However, a stakeholder participating as role r may only interact with those parts of the current state s_{cur} which are visible to this role ($s_{cur}|_r$), leading to changes within this partial

state ($s_{next}|_r$). Consequently, these changes can already sufficiently be derived in a story pattern $SP_{derived} = (s_{cur}|_r, s_{next}|_r)$. As per our definition, the execution of such a story pattern $SP_{derived}$ on an identical state s_{cur} would lead to a state s'_{next} for which $s'_{next} \cong s_{next}$ holds. By default, the activity encoded in this story pattern is assumed to be correct, since it represents the changes as intended and implicitly acknowledged by the participating stakeholder.

For each activity which can be captured by observing a stakeholder participating as a specific role r, the corresponding changes can be attributed to this role. The instance of role r (which exists in the LHS as well as the RHS) becomes the story pattern's *this* object. Per convention, this expresses that the resulting story pattern *belongs* to role r and may only be executed by this role (cf. Section 2.5).

Furthermore, it has to be ensured that no identical story pattern already exists in $\mathcal{SP}$. The story pattern $SP_{derived}$ uses both states reduced to the perspective of the corresponding role r: $s_{cur}|_r$ and $s_{next}|_r$. Thus, each time a state s'_{cur} that is identical from the perspective of r (i.e. $s_{cur}|_r \cong s'_{cur}|_r$) is transformed to a state s'_{next} that is also identical for r (i.e. $s_{next}|_r \cong s'_{next}|_r$), the generated story pattern is rejected. In other words, a new story pattern $SP_{derived}$ is only added to $\mathcal{SP}$ if the following holds: $\nexists SP_i \in \mathcal{SP} : SP_i = SP_{derived}$.

5.1.2. Case Study: Alternatives for the Movie Ticket Sale

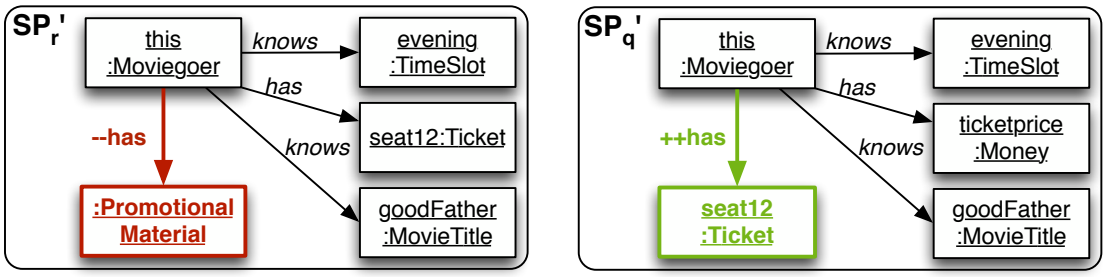
In the cinema example presented in Section 4.1.2, Jane validated the existing story patterns and identified errors from her perspective as a *seller*. Now, John participates as a *moviegoer* to provide new insights into how this group of stakeholders goes through the sale of a ticket. Thus, after loading s_{init} (step *a* in Figure 5.2), John joins into the simulation (step *b*) and can start to interact with the simulator, after the simulator checks which of the story patterns in $\mathcal{SP}$ (Figure 4.4) are applicable. For this initial state, only SP_v is applicable.

Play In of new Behavior

At each state s_{cur} visited during the simulation session, participating stakeholders can choose to interact with the visualization directly in the steps *k, m,* and *n* (Figure 5.2) to fill in the gaps that still remain due to incompleteness. Thus, the ability to play in behavior that was not observed before is essential to cover all scenarios the stakeholders are involved in by exploring the alternatives which may unfold. Starting at s_{init}, the following session unfolds: $s_{init} \xrightarrow{SP_v} s_1 \xrightarrow{SP_a} s_2 \xrightarrow{SP_b} s_3 \xrightarrow{SP_e} s_{10} \xrightarrow{SP_c} s_{11} \xrightarrow{SP'_d} s_6$ (Figure 4.8). In detail, after John starts the visit, he shares the time slot and the title of the movie (SP_v, SP_a, and SP_b, resp.) with a simulated seller. Then, John can either hand over the money for the ticket (SP_e) or the simulated seller may create a ticket (SP_c). Since participants are prioritized, John chooses to execute the proposed story pattern (steps *h, j,* and *e*). Then, since none of the applicable story patterns ($\mathcal{SP}_{applicable} \neq \emptyset$) belong

to the moviegoer John, he has to wait for the simulated seller to create the ticket (SP_c). As a moviegoer, John has already handed over the money. Thus, after the simulated seller has created the ticket (without GUI changes for John who was not affected), the seller hands over the tickets together with promotional material (SP'_d, leading to state s_6 in Figure 4.8). After his GUI is synchronized in step *f*, he can perceive the ticket and acknowledges the correctness of the scenario so far. Thereby, he also validates one of the potentially possible, i.e. *speculative* scenario sequences illustrated in Figure 4.8. Finally, he chooses the proposed story pattern SP_w to leave the *seller* by terminating the visit (steps *h, j, e,* and *f*).

At this point, the simulator can match the current state s_{cur} to the final state s_{term} (cf. Figure 4.3). Since John considers the scenario to be incomplete, he does not acknowledge this final state and continues. So far, John has only relied on story patterns proposed to him by the simulator (i.e. steps *h, j,* and *e*). However, to present his way of completing the scenario, John interacts directly with his GUI (steps *k, m,* and *n*) to throw away the promotional material he just received – "as always", as he remarks. From observing John, who participates as a moviegoer, the simulator can derive the story pattern SP'_r (cf. Figure 5.3a) from the corresponding pair of states. Since this activity takes place in isolation without any further interaction with a seller, the role seller is not affected by it and does not have to approve or validate this behavior. Based on these insights, a requirements engineer can add the current state as an alternative terminal state s'_{term} (similar to s_{term} illustrated in Figure 4.3, except that promotional material is not required) to S_{term}.



(a) A moviegoer was observed *loosing* the advertisement

(b) Moviegoer incorrectly obtains a ticket

Figure 5.3.: An optional (SP'_r) and an incorrect (SP'_q) observation made after observing two moviegoers

Correcting Incorrectly Played In Behavior

Figure 5.3b illustrates behavior that was observed, i.e. played in, from another stakeholder acting as a moviegoer. Again, the simulation starts from s_{init}, in which the

moviegoer is expected to initiate a visit to the seller. If, however, the creation of artifacts is not restricted, the participant may exploit the affordance of being able to create a ticket *himself* while participating as a moviegoer. Consequently, from such an observation, the story pattern SP'_q can be derived. As illustrated in s_{term} (cf. Figure 4.3), this scenario may end if the moviegoer received the ticket and *the seller received the money*. Still, having obtained a ticket, the stakeholder participating as a moviegoer concludes the session and, thus, the session's final state s_{cur} is included to S_{term} as a new potential final state s''_{term}.

However, while it might be tempting for moviegoers to create a ticket without having to pay for it, this observation does not conform to the valid scenarios may which occur at the cinema. Consequently, whenever the behavior in SP'_q is about to be validated in a session, a stakeholder participating as a seller would correctly point out that either she received no money or that the new final state s''_{term} has been reached prematurely before she was able to create the ticket. As a consequence, the underlying formal behavioral specification, i.e. story pattern, may be modified by the requirements engineer to address the stakeholder's critique. Furthermore, by representing the story pattern in natural language using the *NL4SP* approach (cf. Section 2.5), stakeholders may modify the story pattern directly. Alternatively, the story pattern may simply be removed from $\mathcal{SP}$.

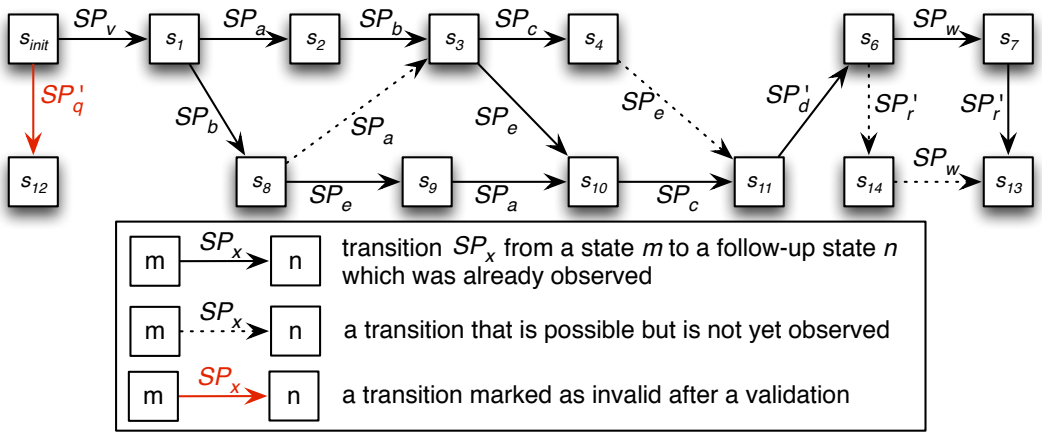


Figure 5.4.: After additional simulation sessions, SP'_r was introduced and SP'_q is marked invalid (continuation from Figure 4.8)

Play In of Alternative Sequences

Based on its precondition, each story pattern can be available at different states during a simulation. Thus, different sequences of states and story patterns between them may exist. However, only a fraction of these sequences is based on observations in prior simulation sessions. The example scenario is quite straightforward: the sequence start-

ing with $s_{init} \xrightarrow{SP_v} s_1 \xrightarrow{SP_a} s_2 \xrightarrow{SP_b} s_3$ was already observed and validated without any stakeholder complaints. Still, the alternative sequence starting with $s_{init} \xrightarrow{SP_v} s_1 \xrightarrow{SP_b} s_8$, but continuing with SP_a was not validated yet (indicated by the dashed line between s_8 and s_3 in Figure 5.4). To allow a moviegoer to validate this alternative and speculative sequence, s_{init} is loaded and the participant initiates a visit to the seller (SP_v), after which the simulation arrives in s_1. Since both applicable story patterns (SP_a and SP_b, as established in step c) belong to the moviegoer, both are proposed (step h in Figure 5.2) and highlighted (step j) to indicate that they have already been observed after SP_v.

If the participant chooses SP_b, the simulation enters s_8 in which SP_a and SP_e can be proposed to the participant (step h). As illustrated in Figure 5.4, only SP_e has already been observed in such a sequence and, correspondingly, is highlighted (step j). By directly proposing speculative alternatives (such as SP_a in this case) to already observed sequences, stakeholders can acknowledge or refute whether the preconditions of the proposed story patterns are correct and whether the new sequence is indeed part of a valid scenario.

If no participant chooses the sequence of SP_b and SP_a himself, the simulator can still execute these story patterns in this sequence to provoke feedback from affected stakeholders participating as seller. This becomes possible as soon as the role of the moviegoer needs to be simulated and, further, can be enforced through the use of strategies.

5.2. Restrictions on Deriving Story Patterns

After a scenario has been observed, the differences between two succeeding states s_1 and s_2 (cf. Figure 4.6) can be interpreted as an action or interaction SP_a (Figure 4.4) of a distinct stakeholder interacting with her visualization of s_1. Actions that can be observed mainly concern the creation, modification or consumption of artifacts such as progress reports or invoices [Gab11]. All artifacts or facts, i.e. pieces of information such as a movie title, can be passed around or shared during those interactions. Through observation of how participants interact with the virtual prototype, they *play in* transitions to states which might not have been reachable from s_{init} solely relying on the story patterns in $\mathcal{SP}$. Apart from exploring individual states which results in corresponding story patterns, participants may also play in completely new scenarios, i.e. alternative sequences of getting from an initial state toward a terminal one. Based on these new states, new story patterns can be derived and added to $\mathcal{SP}$.

The level of detail of what can be described is defined by $\mathcal{D}$. With $\mathcal{D}_{cinema}$ (cf. Figure 4.3), the requirements engineer can build sentences such as: "IF the moviegoer is at the seller (*jane*) and the seller has a ticket (*seat12*), promotional material, and money (*ticketprice*), THEN the seller gives the ticket and the promotional material to the moviegoer" as illustrated in SP'_d (Figure 4.7). However, since only states conforming to $\mathcal{D}$ can be visited during the simulation, it may be required to extend $\mathcal{D}$ to allow participants to reach previously inaccessible states that can be observed within the scenarios under

investigation. Story patterns can only define changes which can be described in the language established by $\mathcal{D}$ (cf. [Küh06]) – if there is no concept or *word* for signature, it cannot be described and, thus, cannot be represented or provided in the GUI. Our approach focuses on allowing stakeholders to achieve what they normally achieve, simply based on the concepts they rely on to achieve it. If stakeholders point out that a specific activity is not possible due to missing concepts in $\mathcal{D}$, it is up to the requirements engineer to extend $\mathcal{D}$ by modeling these concepts correspondingly (cf. use cases in Section 3.1.5). Further, while generic extensions for new types of artifacts are straightforward, unexpected extensions such as a concept representing *locations* (as discussed in Section 3.1.3) may also require a suitable visual metaphor to enable stakeholders to intuitively recognize the implications of such a new concept.

5.3. Merging Preconditions of Equivalent Activities

A stakeholder participating as *moviegoer* can only interact with the part of the current state s_{cur} that is *visible* to him, i.e. the corresponding projection $s_{cur}|_{moviegoer}$. Thus, while the postconditions of an action can be easily deduced from the difference between the state prior to the action and the state the scenario is in afterwards, the preconditions are quite complex if not impossible to infer correctly in an automated fashion. Based on the context of a stakeholder who was observed conducting a specific activity, it is clearly obvious how the context is changed. However, which objects or information were *relevant* for this activity, i.e. which of them *enabled* the stakeholder to execute this activity, cannot be deduced based on the observed context alone.

For instance, while $SP_a''' = (s_{cur}, s_{next})$ represents the resulting story pattern as strictly derived from s_1 and s_2 in Figure 4.6, all other story patterns illustrated in Figure 5.5 are *reduced* versions of the same activity. While $SP_a'' = (s_{cur}|_{moviegoer}, s_{next}|_{moviegoer})$ corresponds to the moviegoer's perspective, SP_a' and SP_a are even less restrictive versions of the same modifications which were observed. In the following, different reductions are discussed which can be performed on a derived story pattern to decrease redundancy among the story patterns. As a reminder, by making a story pattern's precondition less strict, elements removed from LHS also have to be removed from the RHS to keep the encoded activity intact (cf. Section 4.1.2).

Merging Preconditions from Different Scenarios: After $SP_a'' = (LHS_a'', RHS_a'')$ has been derived, alternative scenarios may exist in which the moviegoer shares the time slot without access to any money. Hence, the same modification can be observed in a situation, i.e. state s_{cur} in which not all of the preconditions of SP_a'' are fulfilled. After such an observation, SP_a' would be a suitable reduction of SP_a''', since it covers both alternatives: situations in scenarios which include money as well as scenarios without it. This holds true since, based on the fact that the precondition of SP_a' is a subset of the precondition in SP_a'', SP_a' is applicable in *all* states in which SP_a'' is applicable as well.

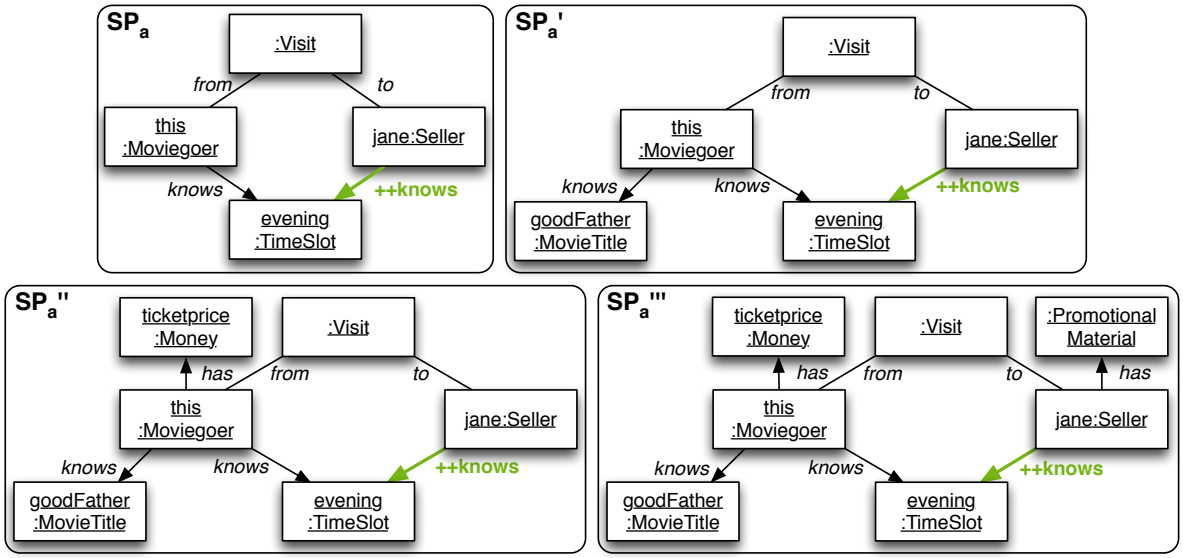


Figure 5.5.: These four story patterns represent the same activity (*sharing a time slot*), although their applicability is different due to their modified preconditions

Minimal Reduction: Based on asking a stakeholder participating as a moviegoer directly *what is essential for him to share the time slot he intends to see the movie with the seller*, the obvious answer would be, that apart from being able to speak with the seller, i.e. *visiting* her, no other precondition is required – except for himself to know the intended time slot. Consequently, $SP_a = (LHS_{min}, RHS_{min})$ is reduced to this minimum, which can only be achieved after either observing such a context $s_{cur}|_{moviegoer}$ during a simulation, or asking the stakeholders which parts of the preconditions they require access to. As pointed out by Alshanqiti et al. [AHK13], such a minimal version of the same rule may also be merged from multiple observations – successful and failed ones as indicated by participants during validations. Based on feedback of stakeholders, preconditions can further be refined, e.g. by introducing NACs to ensure that specific story patterns are not applicable in distinct states during the simulation. The refinement or modification of a story pattern may even be delegated to the stakeholders themselves by representing the story pattern in NL (cf. Section 2.5).

Each time preconditions of a story pattern SP are modified, its applicability changes. Consequently, stakeholders need to validate the resulting preconditions in subsequent simulation sessions.

5.4. Chapter Summary

In this chapter, we extended our simulation approach to empower stakeholders to complete each other's scenarios through the interactive user interface of the virtual proto-

types. While story patterns are derived from observing participants, the preconditions of these automatically generated story patterns still need to be validated. Specifically, the validation supports the identification of problems such as incorrect observations, erroneously merged preconditions, and inconsistencies between different stakeholders enacting the same role.

Up until now, we implicitly relied on the assumption that a participant is always capable of continuing the scenario – eventually, after several iterations. In situations where the stakeholders' availability is limited, this is not always feasible, since it cannot be assumed that a participant can always fill-in the blanks which remained from earlier sessions. This simplifying assumption is overcome by the extension introduced in the following chapter.

6. Decoupled Completion and Correction of Scenarios

The requirements engineers start by eliciting scenarios from individual stakeholder perspectives. After combining such fragmented instance scenarios into a consistent overall scenario model, they validate this scenario to exclude elicitation errors and check whether they have covered all alternatives. Then, these activities continue iteratively with additional elicitation activities, updates of the modeled scenarios, and subsequent validation activities until the result is agreed upon. The resulting global scenario is the crucial element to ensure that a consistent understanding of the different stakeholder perspectives can be established. Of course, the elicitation and validation of scenarios requires less effort if all stakeholders are involved simultaneously. If all participants directly comment on whether they agree with the statements of other stakeholders, the requirements engineer might obtain a commonly agreed-upon scenario model directly within an elicitation session, similar to Luebbe and Weske's approach [LW11]. However, due to scheduling and resource constraints, such a setting is usually less efficient compared to elicitations with individual stakeholders [SMK+09] if not unfeasible. Also, experience shows that in case of group meetings social effects can result in suppressed opinions and observations of stakeholders positioned lower in the hierarchy [Mur11]. Furthermore, the stakeholders who participate are sometimes chosen based on who is noncritical for the daily work to continue without interruptions [ARE96]. To limit such effects, techniques that permit all stakeholders to be involved in the elicitation and validation without needing to be present in person at the same location and at the same time are required.

So far, the presented approach does not support stakeholders in overcoming situations in which their next steps depend on how other roles respond or interact with them during the play-in and exploration of new scenarios. Therefore, participants can only complement each other's activities if applicable counterparts to their interactions[1] (i.e. plausible continuations of the scenario) have already been observed and can be replayed. Since the simulator cannot know how to react if no suitable response has been observed, stakeholders run into dead ends during their elicitation sessions (referred to as *stalemates*). Thus, the simulation concepts presented so far required that the stakeholders involved in this interaction to participate either simultaneously or multiple times to be able to complement each other. Since arranging a meeting of all involved

[1] In accordance with Dirgahayu et al. [DQS10], an *interaction* is considered "an action performed by multiple entities in cooperation to establish a result that is acceptable by all involved entities".

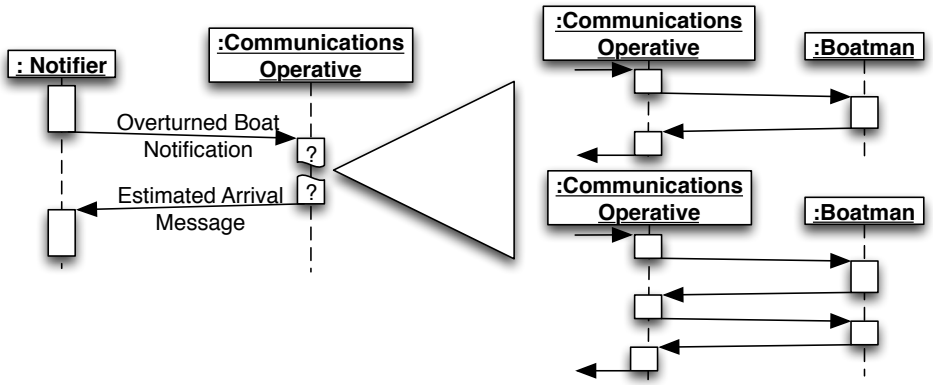


Figure 6.1.: The scenario initiated on the *left* (a Notifier calling in an emergency) is incomplete and the Notifier cannot know which of the possible continuations (*right*) will occur

stakeholders to elicit the new scenarios in one simulation session is too complex and time-consuming, each stakeholder night have to participate in multiple sessions, waiting for other stakeholders to play in their continuations for the scenario. This can lead to numerous sessions, especially for stakeholders working in coordinating roles. Only by being able to further decouple individual stakeholders during the elicitation and validation of their interactions, can a complete, systematic, and *feasible* elicitation of all scenarios be ensured.

This chapter extends our simulation approach in a way that it overcomes stalemates by bridging the unknown fragments of the scenario and, thereby, can reduce the number of necessary sessions. By enabling participants to explicitly express their expectations of what changes for them after interacting with other stakeholders in form of partial states, other participants can continue playing in their parts of the scenario, thereby fulfilling these expectations without requiring additional elicitation sessions. Monitoring the execution of the simulation as described in [Teu11] enables this extension of the simulator to recognize the fulfillment of such expectations to identify suitable continuations of incomplete scenarios. Consequently, the results of multiple simulation sessions can be combined automatically based on fulfilled expectations.

Since this chapter deals with challenges arising from decoupling stakeholders, it is assumed that only one stakeholder participates in this *decoupled* mode of the simulation throughout this chapter. This chapter is based on our publication [GHG12a] and structured as follows: Section 6.1 discusses our approach on how stakeholders can describe their *expected continuations* by simply answering three questions to define *triggers* and two questions for *follow-up actions*. Then, Section 6.2 presents a case study based on practical examples elicited from a lifeguard service during a requirements engineering seminar we conducted [GHPG13]. Finally, a summary is presented in Section 6.3.

6.1. Concept

In each simulation session, each participating stakeholder has a unique perspective on the current state s_{cur} of the simulation. After each activity of any role, each participant might be affected by the result. Still, only some of these activities and their results are even visible to the role in which this stakeholder participates. Consequently, every time a participant is affected by a change of the current state of the simulation, this change has to be reflected in the stakeholder's visualization (step f in Figure 6.2). This visualization illustrates the current state s_{cur} of the simulation, reduced to what a participant's role $Role_T$ is able to perceive ($s_{cur}|_{Role_T}$).

After the simulator has reached a state s_{sm} in which no story pattern is applicable (step c returns $\emptyset$), the stakeholder participating as $Role_T$ is expected to play in alternative behavior. However, s_{sm} is considered to be a *stalemate* if $Role_T$ arrived in s_{sm} after initiating an interaction with another role $Role_{Req}$ and must wait for the execution of an activity of $Role_{Req}$ which cannot be simulated since it was not yet observed. Generally, a stalemate can only be overcome if the requirements engineers gather the other side of the interaction. Similar to a black box, we can only assume how $Role_{Req}$ continues after being triggered by $Role_T$. Still, while $Role_{Req}$'s activities are not yet known, $Role_T$ can describe most of the possible outcomes, i.e., how he is affected by the result, based on experience. Similar to a jigsaw puzzle, many pieces of information exist, however, only a few of them are required to complete individual scenarios. Thus, it is essential that individual stakeholders do not have to give up at the first stalemate, but are able to continue to describe their expectation(s)[2] as well as their follow-up actions of how they continue after the expectation is fulfilled. Since an individual stakeholder cannot know how the overall state s_{sm} of *all* involved roles changes in between, he can only specify his perspectives of the respective states. Thus, an expectation can only be a partial state $s'_{sm}|_{Role_T}$ describing the changes the stakeholder expects to perceive based on his individual perspective ($s_{sm}|_{Role_T}$).

	Question for Stakeholder	*Answer*
Q_1	Who did you interact with?	$Role_{Req}$
Q_2	Who, if not [$Role_{Req}$], do you expect to get an answer from?	$Role_{Resp}$
Q_3	What do you expect to change [based on $s_{sm}|_{Role_T}$]?	$s'_{sm}|_{Role_T}$

Table 6.1.: By answering these questions, a stakeholder describes which interactions usually occur, who is involved, and how they affect him

If a stalemate occurs during an elicitation of a scenario, a participant can still answer the questions in Table 6.1. Q_1 provides the requirements engineers with the information

[2] The results of "what customers expect when contacting the service provider, by the telephone for example, with this being contrasted with similar [expectations] of in-person visits or email correspondence" (cf. *Expectation Maps* in [SS12b]).

of who to talk to next to get closer to the completion of the incomplete scenario. Possible answers are the roles that are defined in the domain model $\mathcal{D}$. Only a stakeholder identified as role $Role_{Req}$, i.e., someone who usually receives $Role_T$'s request, knows how to continue. Still, in some cases, $Role_{Req}$ is not the role that is expected to respond. Furthermore, if multiple responses, which may originate from different roles, need to be distinguished, the (otherwise optional) second question Q_2 provides information on whom else might provide a result which $Role_T$ expects.

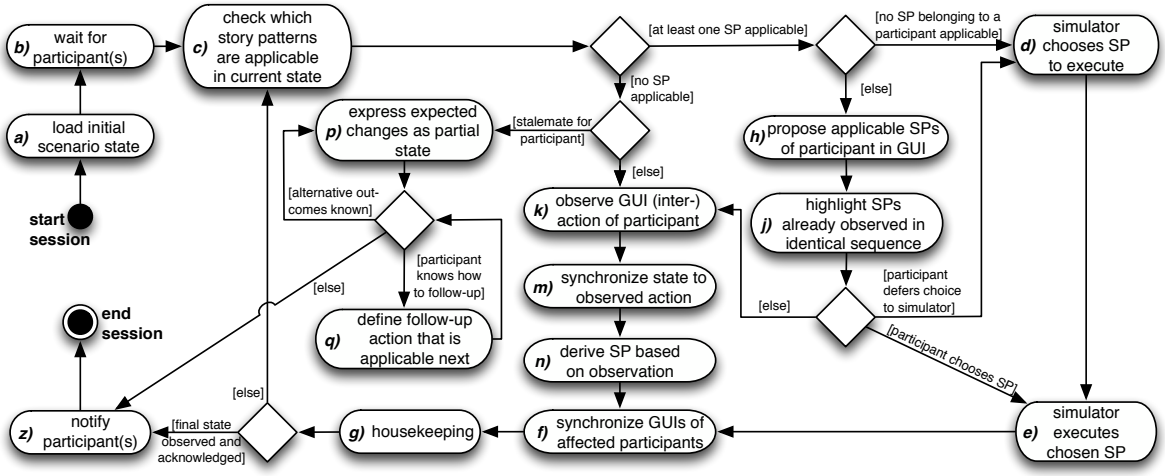


Figure 6.2.: Complete simulation loop which supports *play out*, *play in*, and *triggers*

The simulation has a specific state s_{sm}, in which the stalemate occurred. In this state, an interaction has been started that results in a change for $Role_T$ – although he does not know how anyone else might be affected as a side effect, the stakeholder can still describe what changes for him, i.e. answer question Q_3. To both simplify and structure this description, the participant's perspective $s_{sm}|_{Role_T}$ is represented in NL using the NL4SP approach [GEHG13] and provided as a template for the required response to Q_3. The participant can reshape this partial state to reflect the changes they expect to be affected by. Thus, based on the differences between what the participant currently perceives ($s_{sm}|_{Role_T}$) and the participant's answer to Q_3 ($s'_{sm}|_{Role_T}$), an expectation is defined.

6.1.1. Capturing Different Stakeholder Expectations as Triggers

As illustrated in Figure 6.1, the requirements engineers have to deal with individual perspectives as well as handovers. It happens quite often, that a stakeholder $Role_T$ telling his story cannot continue after he hands over a critical artifact, requests information or starts any other form of interaction with another stakeholder $Role_{Req}$. Since $Role_T$ does not know what Desai et al. [DMCS05] refer to as local and usually *private policies*, which

dictate how $Role_{Req}$ acts or reacts in a specific situation, we can never be completely sure what happens without talking to a corresponding stakeholder. We refer to such a potential dead end during the elicitation as a *stalemate* – without information on how another role continues the scenario, requirements engineers and the stakeholder can only guess what happens next. To enable participants to describe their expectations using triggers, the simulation loop was extended with the step p as illustrated in Figure 6.2.

p) Express Expected Changes as Partial State: A *trigger* is a tuple (s_{sm}, $Role_T$, $Role_{Req}$, $Role_{Resp}$, $s'_{sm}|_{Role_T}$). It contains the *stalemate* state s_{sm} as it occurred during the simulation. Further, the role of the participant who defined the trigger ($Role_T$) is included. Additionally, to resolve the trigger, it is essential to know which role is expected to continue the scenario and which role is expected to interact with $Role_T$ next ($Role_{Req}$ based on Q_1 and $Role_{Resp}$ based on Q_2, respectively). Finally, the partial state $s'_{sm}|_{Role_T}$ that $Role_T$ expects to observe afterwards is included as well, based on Q_3. This expectation can be defined using the NL4SP approach (cf. Section 2.5).

Often, there is an information asymmetry between different roles. Thus, a stakeholder cannot know what a decision that affects her is based on. Still, by enumerating the different possible continuations, all different scenarios can be captured systematically, as opposed to only one at a time. Consequently, based on the current state as perceived by the participant ($s_{sm}|_{Role_T}$), other alternatives can be described as well by simply answering Q_2 and Q_3 again for the alternative expectations. Hence, from each stalemate s_{sm}, multiple expectations can be defined as part of different triggers.

6.1.2. Capturing Stakeholders' Follow-Up Actions

A *follow-up action* f is an activity of a role which is expected to apply if a specific precondition is fulfilled. It is characterized by a pair of states, the first being a precondition ($s_F|_{Role_T}$), which has to be fulfilled to execute the changes leading to the second ($s'_F|_{Role_T}$). Questions similar to those necessary to define triggers can be used to elicit follow-up actions. The corresponding questions are presented in Table 6.2. While Q_A establishes what has to be fulfilled for the participant to consider the activity, Q_B contains the postcondition which is reached after the activity is executed.

	Question for Stakeholder	*Named*	*Part of Story Pattern*
Q_A	When do you become active [again]?	$s_F|_{Role_T}$	Left-Hand Side
Q_B	How do you continue after $[s_F|_{Role_T}]$?	$s'_F|_{Role_T}$	Right-Hand Side

Table 6.2.: By answering these questions, stakeholders can specify a distinct partial state (LHS) and how they follow up on it (RHS of a resulting story pattern)

Moreover, since both partial states in a follow-up action $f_i = (s_F|_{Role_T}, s'_F|_{Role_T})$ can be considered as precondition and postcondition, a follow-up action f_i is equivalent

to a story pattern $SP_i = (LHS_i, RHS_i)$ if $LHS_i \cong s_F|_{Role_T}$ and $RHS_i \cong s'_F|_{Role_T}$. Consequently, each follow-up action f_i leads to a story pattern SP_i which can be applied as soon as the current state s_{cur} of the simulation contains a match for $s_F|_{Role_T}$, i.e. the precondition of SP_i. Since we know which role the participant enacted, the story pattern belongs to this specific role.

q) Define Follow-Up Action that is Applicable next: Since a trigger $t_i = (s_{sm}, Role_T, Role_{Req}, Role_{Resp}, s'_{sm}|_{Role_T})$ has to be resolved for the participant playing as $Role_T$ to follow up, the simulation needs to be in a state in which the expectation of this trigger is fulfilled. Consequently, if a follow-up action is defined directly after a trigger, the trigger's *expected* partial state ($s'_{sm}|_{Role_T}$) can automatically be offered as the default answer to the follow-up action's question Q_A, i.e. $s_F|_{Role_T} := s'_{sm}|_{Role_T}$. Combined with the follow-up state, i.e. the answer to Q_B, this leads to the follow-up action $f_i = (s'_{sm}|_{Role_T}, s''_{sm}|_{Role_T})$. As explained above, these two partial states can be used to derive a story pattern SP_i which captures the intention of what the participant would do after $Role_{Resp}$ of t_i responded *as expected*. As opposed to triggers, the definition of additional follow-up actions starts with the postcondition $s'_F|_{Role_T}$ of the prior follow-up action as *pre*condition for the next; after $f_i = (s_F|_{Role_T}, s'_F|_{Role_T})$, f_2 would be $(s'_F|_{Role_T}, s''_F|_{Role_T})$. This allows stakeholders to describe sequences of activities which complement one scenario at a time. Again, the NL4SP approach may be used to enable stakeholders to modify or define the required partial states.

6.1.3. Resolving Triggers Systematically

After the requirements engineer has elicited an incomplete collaborative scenario ending in a stalemate s_{sm} and at least one trigger $t_m = (s_{sm}, Role_T, Role_{Req}, Role_{Resp}, s'_{sm}|_{Role_T})$, the next stakeholder to talk to is already predetermined. To complete this scenario, a stakeholder of the corresponding role $Role_{Req}$ needs to participate to continue the interaction with $Role_T$. The simulation starts by loading the state s_{sm} for $Role_{Req}$, so that the simulation can continue directly from the last complete state of the incomplete scenario. The visualization of s_{sm} corresponding to $Role_{Req}$'s perspective $s_{sm}|_{Role_{Req}}$ visualizes the last interaction with $Role_T$ in such a way that the participating stakeholder is able to identify which scenario the requirements engineers are currently interested in. By explicitly loading s_{sm}, inconsistencies between different triggers and their continuations can be avoided, since all follow-up activities are considered direct responses leading towards the expectation $s'_{sm}|_{Role_T}$.

All triggers that are collected along the way are stored next to the terminal states in S_{term}. The follow-up actions, on the other hand, are stored next to the derived story patterns. To resolve a trigger, the simulator simply checks in each loop (step c) whether the expected outcome of an interaction ($s'_{sm}|_{Role_T}$) can be matched in the current state s_{cur} of the simulation. After a trigger has been fulfilled, a sequence has been found which potentially complements the incomplete scenario of $Role_T$. Then, potential follow-up

actions the participant whose trigger was fulfilled may have defined are checked, as are all other story patterns in step *c*. If the simulator decides how to continue in step *d*, follow-up actions are executed with priority, since these actions allegedly belong to the same scenario. Thereby, the simulator enforces the completion of such fragmented scenarios. Furthermore, specific strategies that score whether any of the expected changes are reachable within a look ahead can be formulated (cf. *Scoring Individual States* in Section 4.3.1). This bridges any unknown fragments of the collaborative scenario – $Role_T$ may continue to fill-in the blanks between s_{sm} and $s'_{sm}|_{Role_T}$ later on.

Based on this algorithm, scenarios are completed step by step by different stakeholders in subsequent, remote sessions, which decouple them temporally and locally. The possibility remains that no state fulfilling the expectation (i.e. $s_{cur} \supseteq s'_{cur} \cong s'_{sm}|_{Role_T}$) can be reached – even after multiple sessions of the role that is expected to reply. In this case, the requirements engineer has to be notified and two options exist: either talk to $Role_{Resp}$ to ask, e.g., *what needs to be true for* $Role_{Resp}$ *to behave as expected* or talk to $Role_T$ to ensure that the described expectation is correct.

6.2. Case Study: Notifying a Lifeguard Service

This section illustrates the concepts of expected continuations in triggers and follow-up actions using a case study elicited from a lifeguard service (cf. [GHPG13]). For this case study, Figure 6.3 illustrates an abridged version of the domain model $\mathcal{D}_W$. As can be seen in $\mathcal{D}_W$, boats can only be seen by roles which are at the same location. (cf. Figure 6.3). The scenario starts with s_{init}[3] as illustrated in Figure 6.4 and covers the elicitation of the communication between a bystander notifying the lifeguards (referred to as *Notifier*) about an emergency, the corresponding communications operative (referred to as *ComOp*), and a *Boatman*. Three consecutive sessions are necessary: starting with the notifier in the first session (Section 6.2.1), the communications operative continues (Section 6.2.2) before the boatman finalizes the scenario in the third session (Section 6.2.3). All of these sessions are displayed in Figure 6.5.

6.2.1. Session 1 – Notifier

To elicit the emergency notification scenario, a Notifier participates in the simulation. In his visualization of s_{init} (Figure 6.4), the Notifier can recognize the overturned boat which is at his location and which he needs to notify the lifeguard service about. As illustrated in Figure 6.1, the Notifier starts the notification scenario by sending an *OverturnedBoatNotification* to the ComOp. This notification leads the simulation into the state s_m (cf. Figure 6.7). However, the Notifier cannot continue, since he does not know

[3] This state is also the initial state s_{init} referred to in the sequence diagrams throughout this chapter. In the following, all state labels in sequence diagrams refer to complete or partial states represented by such object diagrams.

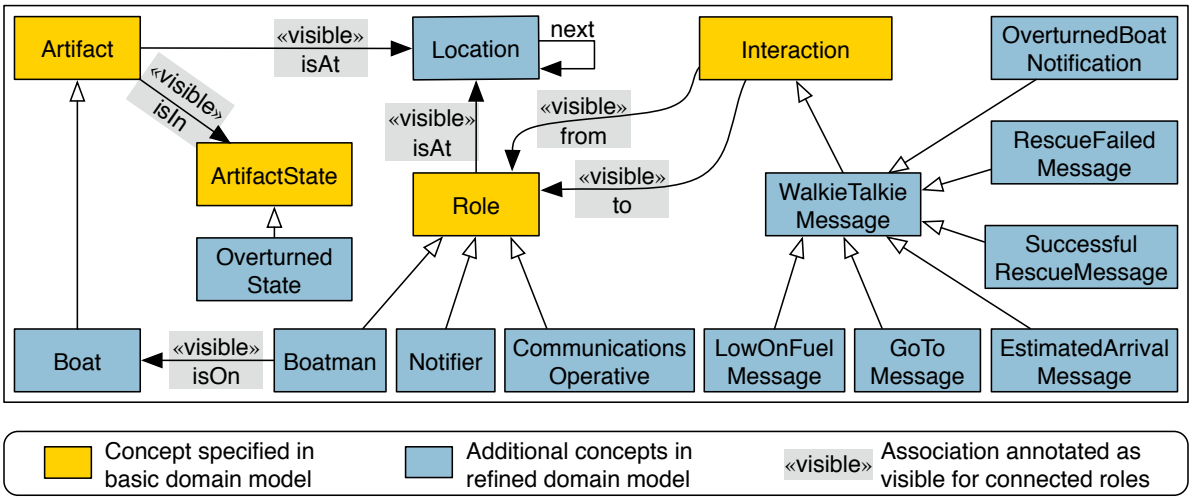


Figure 6.3.: (Abridged) domain model $\mathcal{D}_W$ elicited from a lifeguard service

for sure what happens next. Thus, a stalemate occurs in s_m and the stakeholder participating as Notifier is asked to describe his expectations on how another stakeholder he interacted with might respond or, more generally, how his context may change. Consequently, a trigger can be created based on $s_m|_{Notifier}$. Firstly, the Notifier provides the last interaction partner (Q_1 in Table 6.1). For the Notifier, this would be the ComOp who he notifies about the emergency (Figure 6.1). After the ComOp passed on the information, the Notifier expects to hear from her again – consequently, Q_2 would be answered the same. Then, $s_m|_{Notifier}$ is represented in NL to enable the stakeholder to express how he expects to be affected, thereby answering Q_3. The Notifier expects to receive an estimation on when help will arrive. Thus, using the vocabulary already established as part of the domain model $\mathcal{D}_W$, this expectation can be described explicitly. The Notifier expects $s_m|_{Notifier}$ and $s'_m|_{Notifier}$ to be identical, except that he has to receive an *EstimatedArrivalMessage* from a ComOp. Whether a boat already departed to his location, or whether another emergency happened somewhere else is not clear to the Notifier, providing none of those things are visible to him. The expected follow-up state can be represented and described in different ways to be suitable for stakeholders.

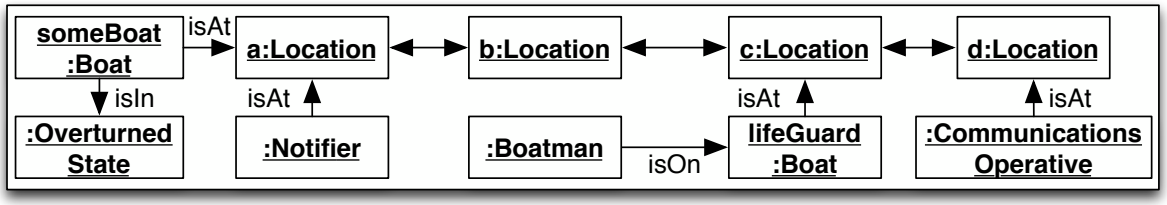


Figure 6.4.: An initial state s_{init} for the lifeguard service scenarios

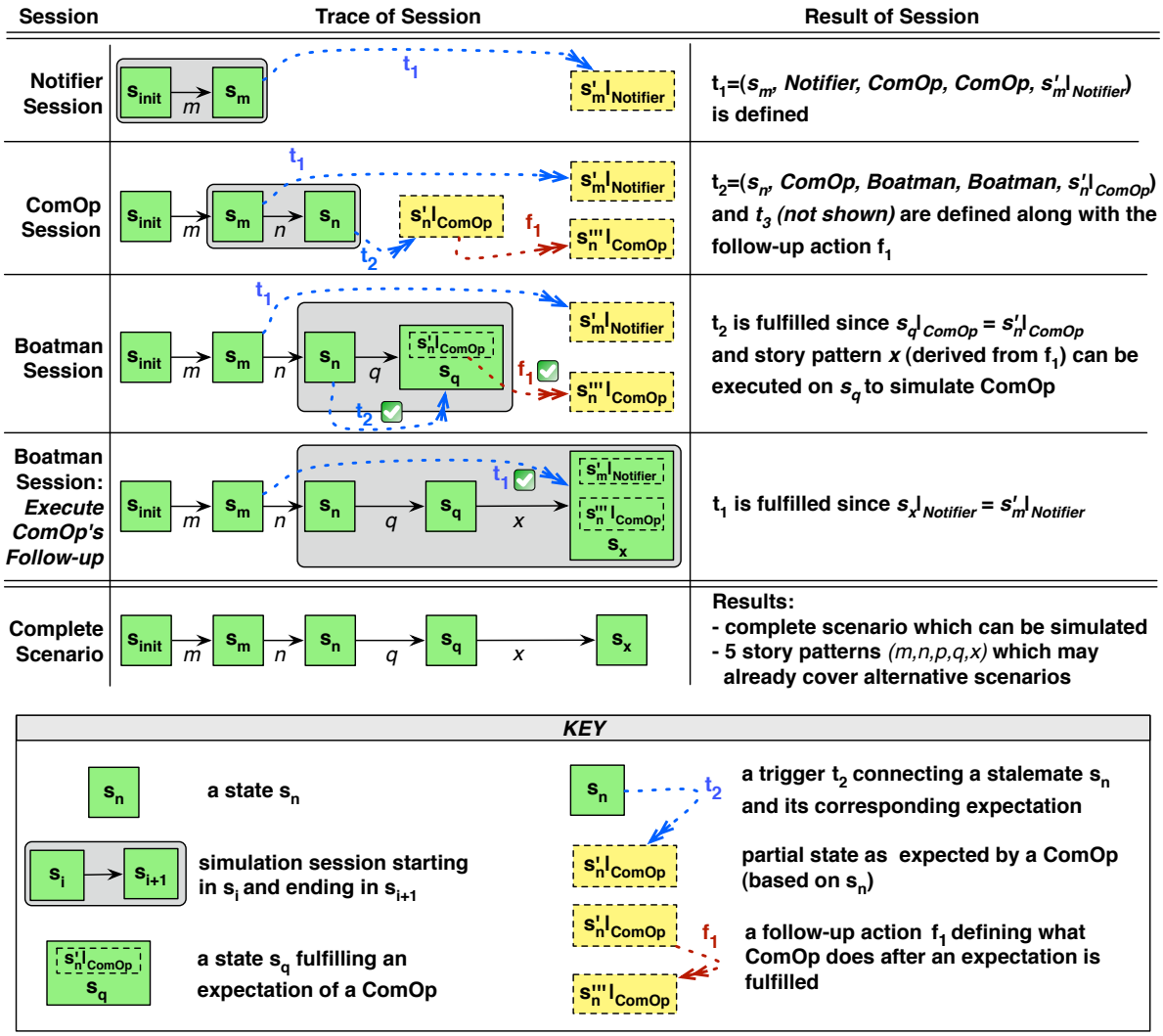
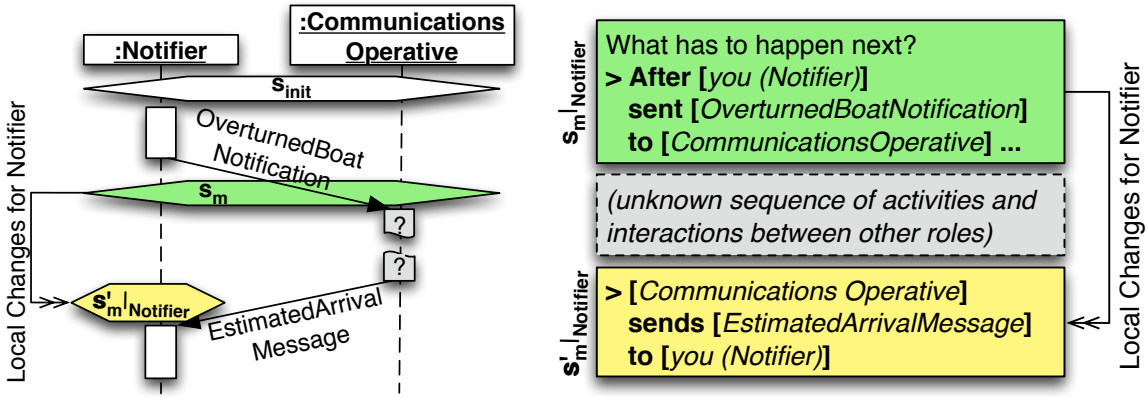


Figure 6.5.: After only three stakeholder sessions, the scenario has been completed with two triggers and one follow-up action (adapted from [GHG12a])

While Figure 6.7 (*right*) illustrates this expectation as a partial state in an object diagram, Figure 6.6 (*right*) presents a natural language representation that can easily be understood and modified: *[CommunicationsOperative] sends [EstimatedArrivalMessage] to [you (Notifier)].*

Finally, the Notifier's answers result in the trigger $t_1 = (s_m, \mathit{Notifier}, \mathit{ComOp}, \mathit{ComOp}, s'_m|_{\mathit{Notifier}})$, as illustrated in Figure 6.7. Furthermore, the Notifier may also describe additional expectations such as the fact that he expects a lifeguard boat to arrive at his location as illustrated in Figure 6.8b. Afterwards, his session is concluded.



(a) *Notifier* knows how he informs the *ComOp* (*green*)

(b) *Notifier* can validate his expectations, i.e., how he is affected (*yellow*), in a NL representation

Figure 6.6.: While the *Notifier* knows how he informs the *ComOp* (*green*), he does not know how the *ComOp* continues this scenario (*gray*); still, the *Notifier* can formulate his expectations, i.e., how he is affected (*yellow*), in NL (*right*)

6.2.2. Session 2 – Communications Operative

To resolve the trigger t_1 that Notifier created in the first session (1^{st} row in Figure 6.5), a ComOp is required for the next session. As the last interaction partner of Notifier, it is implied, that a ComOp knows how to continue. Thus, the stalemate s_m of the trigger (illustrated in Figure 6.7) is initialized for the session and the participating ComOp stakeholder receives Notifier's notification of an overturned boat in her corresponding visualization $s_m|_{ComOp}$.

Sometimes, $Role_{Req}$ is also the responding role $Role_{Resp}$, which can answer $Role_T$'s interaction directly to fulfill $Role_T$'s expectation with a suitable response. In the case of t_1, however, ComOp cannot yet fulfill Notifier's expectation ($s'_m|_{Notifier}$). As always, the ComOp continues by starting an interaction with a Boatman by sending a *GoToMessage*, thereby bringing the simulation into the state s_n (2^{nd} row in Figure 6.5). At this point, the ComOp cannot continue to play in what needs to be done, since a stalemate s_n is reached in which she cannot deterministically predict how the Boatman will react, since two different scenarios are possible.

After the ComOp's *GoToMessage* sending a Boatman to another location in response to the notification, ComOp expects that the Boatman responds with an *EstimatedArrivalTime*, as prescribed by their protocol. Consequently, the ComOp defines a trigger $t_2 = (s_n, ComOp, Boatman, Boatman, s'_n|_{ComOp})$ expecting this message. While the ComOp might usually get an *EstimatedArrivalMessage*, she might also be confronted with a *LowOnFuelMessage*, indicating that the boat needs to refuel first (illustrated in Figure 6.11). Of course, the ComOp knows that all boats are fueled up at the beginning of each weekend. However, she does not know how much gas each boat may have left after

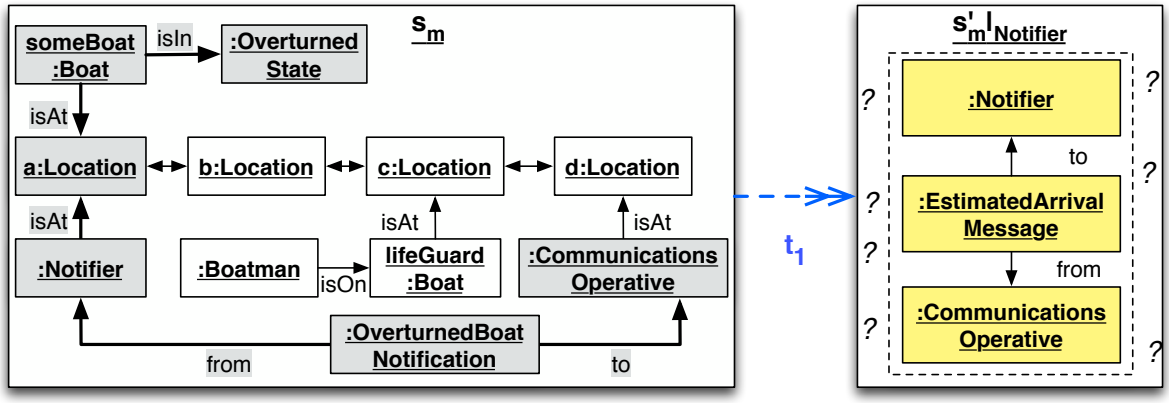
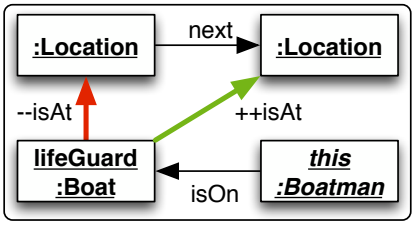


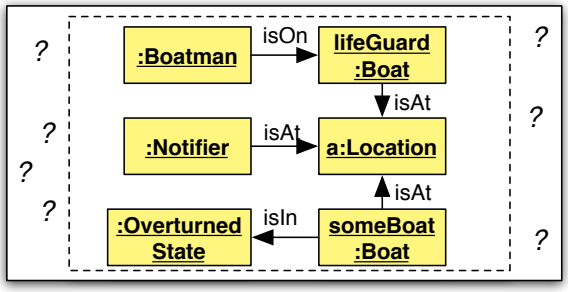
Figure 6.7.: Based on s_m (*left*, Notifier's visibility is highlighted in *gray*), the Notifier can describe changes he expects as a partial state $s'_m|_{Notifier}$ (*right*) in $\mathcal{D}_W$'s vocabulary, leading to trigger t_1 (adapted from [GHG12a])

several hours of service since this information is only available to each respective Boatman. Thus, although a ComOp knows both possible outcomes, she cannot know which one she will be confronted with, since she has no access to the information required for this decision.

After ComOp's expectations have been captured in t_2, the participating stakeholder can still describe how she as a ComOp would continue after her expectation ($s'_n|_{ComOp}$) is fulfilled. Consequently, the partial state as expected in t_2 is presented to the stakeholder, either in an interactive visualization or in a textual representation as provided by NL4SP. Based on this expectation as answer to Q_A, the stakeholder is able to specify the differences resulting from her follow-up actions. For this example, the answer to Q_B (cf.

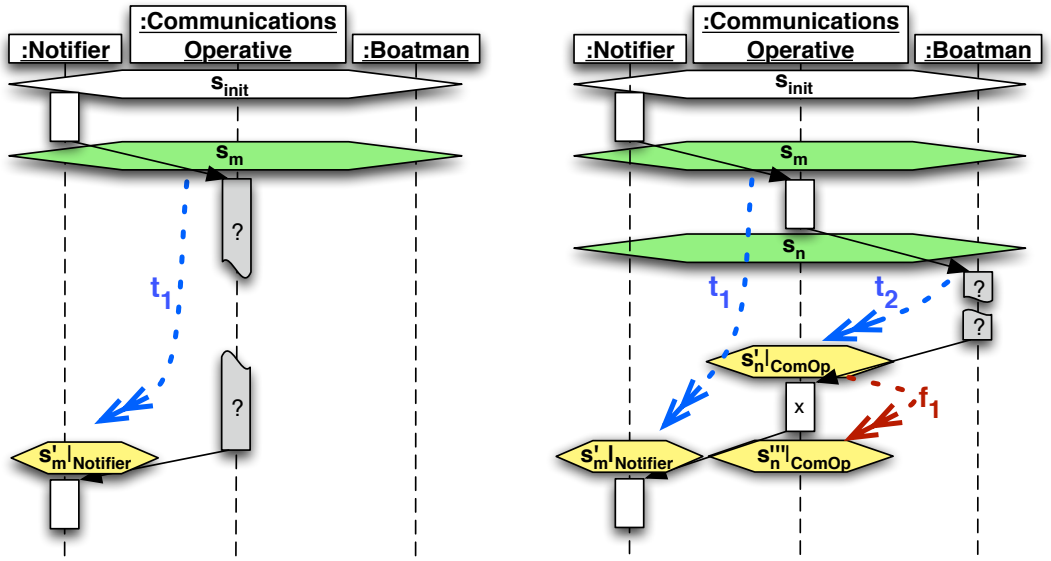


(a) Going to the next location



(b) Eventually, the Notifier expects a lifeguard boat to arrive at his location

Figure 6.8.: Only by moving from one location to the next *(a)*, can a Boatman eventually fulfill a Notifier's expectation *(b)*

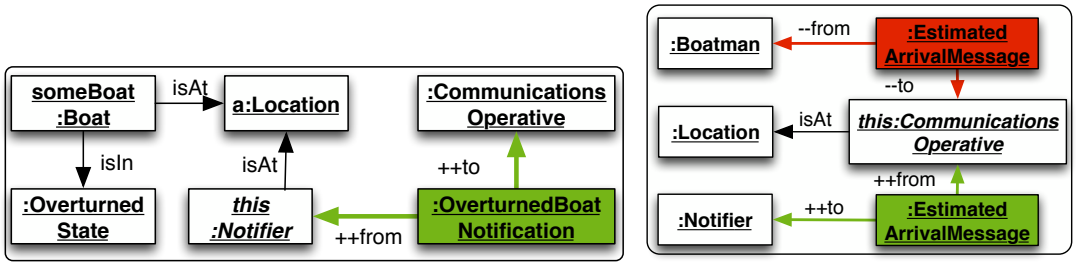


(a) Incomplete scenario after a Notifier session (b) Same scenario after a ComOp follows up

Figure 6.9.: The expectation described in t_1 *(a)* might be fulfilled after t_2 has been resolved and follow-up action f_1 was executed *(b)*

Table 6.2) would be: *"[I] send the Notifier an EstimatedArrivalMessage"* $(s'''_n|_{ComOp})$. Thus, the follow-up action $f_1 = (s'_n|_{ComOp}, s'''_n|_{ComOp})$ leads to the story pattern SP_x (cf. Figure 6.10b) of how the ComOp would continue as soon as the expectation defined in t_2 has been fulfilled.

From s_n, as for most stalemates, multiple continuations are possible from ComOp's point of view. Consequently, the *LowOnFuel* alternative is also based on the stalemate s_n and can be defined as trigger $t_3 = (s_n, ComOp, Boatman, Boatman, s''_n|_{ComOp})$ (cf. Figure



(a) Notifying the life guard service (b) SP_x based on follow-up action f_1

Figure 6.10.: After the ComOp was informed by a Notifier *(a)*, the ComOp defines a follow-up action of how she continues which leads to SP_x *(b)*

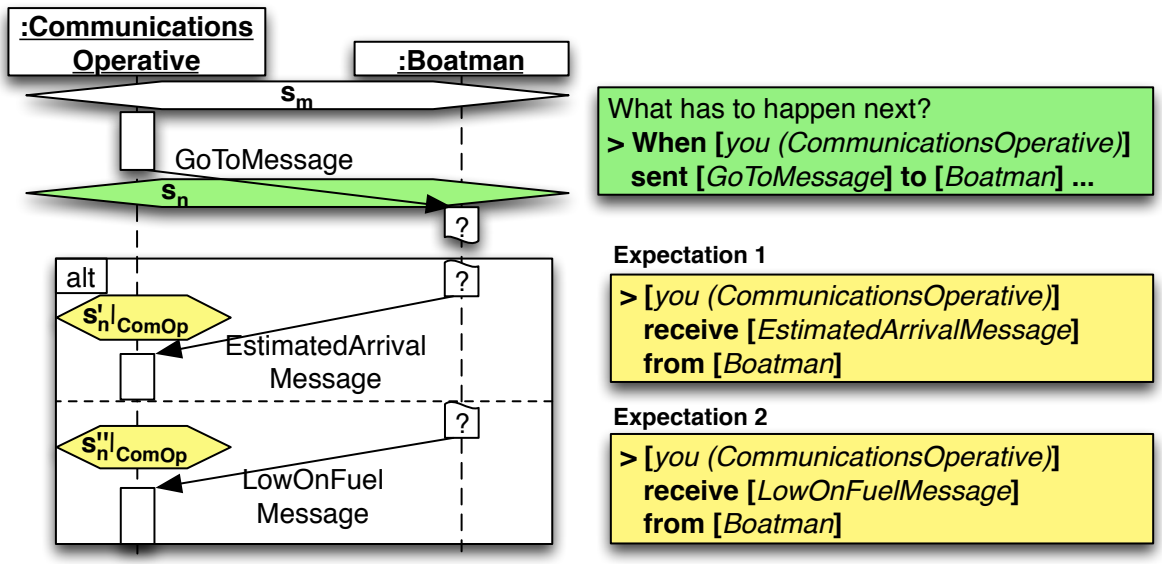


Figure 6.11.: Both expectations for how a Boatman may react to a *GoToMessage* as experienced before and, therefore, expected by a ComOp

6.11). After these triggers and the follow-up action have been defined, this session is concluded (as illustrated in the 2^{nd} row of Figure 6.5).

6.2.3. Session 3 – Boatman

As specified in t_2 and t_3, the next role to talk to is Boatman, who is in both cases expected to continue from s_n. After the participant reviewed s_n in his visualization, he responds with an estimated arrival time (3^{rd} row), thereby leading the simulation into state s_q in which his role Boatman fulfills ComOp's expectation as defined in t_2.

In s_q, ComOp's follow-up action f_1 (represented by SP_x) is applicable, since its precondition is coincidentally identical to t_2's postcondition ($s'_n|_{ComOp}$). Consequently, by fulfilling t_2, the ComOp can be triggered which, in this case, allows the simulator to execute SP_x with priority over any other applicable story patterns. This leads the simulation into the follow-up state s_x as illustrated in the 4^{th} row in Figure 6.5. More importantly, the initial expectation of Notifier can be matched since the expected answer was provided through ComOp's follow-up action ($s_x \supseteq s'_m|_{Notifier} \cong s'''_n|_{ComOp}$, cf. Figure 6.12).

Even though f_1 was executed and t_1 was fulfilled, the boatman still participates. By doing so as he was told (*GoToMessage*), he can be observed going from one location to the next as illustrated in Figure 6.8a. Thus, eventually, he arrives at the Notifier's location which, in turn, fulfills the Notifier's expectation illustrated in Figure 6.8b.

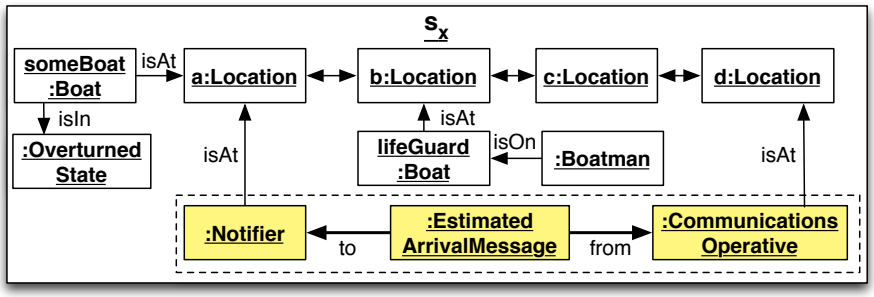


Figure 6.12.: After ComOp is simulated using the story pattern SP_x, the simulation is in state s_x with $s_x \supseteq s'_m|_{Notifier} \cong s'''_n|_{ComOp}$ (fulfilling the highlighted expectation in Figure 6.7)

6.2.4. Resulting Scenario

After these three sessions, the requirements engineers systematically observed and derived all the story patterns required to execute and simulate the complete notification scenario (5^{th} row in Figure 6.5). Since all expectations have been fulfilled, an agreed upon scenario has been established, since all stakeholders were affected as expected and nobody pointed out errors as discussed in Section 4.2. Furthermore, the requirements engineers can continue by systematically covering the alternatives, e.g., *what has to happen when a Boatman fulfills the expectation described in* t_3*?* Of course, the scenario may even continue, since, apart from seeing a lifeguard boat at his location (Figure 6.8b), the notifier may also expect to see that the overturned boat is returned in an upright position (i.e. its default state).

The total number of sessions required to elicit a complete scenario is related to the number of stalemates stakeholders run into during elicitations, since each stalemate may require an additional session to be resolved before $Role_T$ may continue. Generally, the number of stalemates per role can vary strongly. Especially for the role ComOp in the discussed example, at least ten interactions with other roles need to be elicited for each scenario. However, by being able to express her expectations and triggers intuitively, the ComOp is now able to defer the completion of all scenarios to other stakeholders later on. Consequently, instead of ten sessions to overcome ComOp's stalemates only, ComOp can express all her contributions to the collaborative scenarios in just one session. Decoupled from her, the other stakeholders can then complement the scenarios accordingly.

6.3. Chapter Summary

In this chapter, we presented an approach that reduces the effort necessary to initially elicit and validate unknown collaborative scenarios. Thereby, the number of sessions necessary to elicit a stakeholder's actions within a fragmented scenario can be reduced.

In the case of stalemates, stakeholders can express their expectations on interaction results in the form of partial states. Based on these partial scenarios, stakeholders are then able to play in continuing behaviors decoupled from one another. Our simulator synthesizes the captured individual perspectives to obtain the complete scenarios, thereby overcoming the inherent fragmentation of different perspectives. We discussed how this technique extends our model-based validation approach in order to be applicable for model-based elicitation, too. In this chapter, we also illustrated how the discussed extension can be applied in order to systematically complete fragmented scenarios of collaborating stakeholder groups within a lifeguard service.

Without the inclusion of expected continuations as triggers and follow-up actions, the requirements engineer would have to go back and forth between two or more stakeholders for each interaction. By systematically eliciting such interactions using triggers and follow-up actions, on the other hand, the total number of elicitation sessions no longer depends on the number of interactions and stalemates, but on the number of roles and the stakeholders' ability to express their expectation. In this case, our approach can end up with only one elicitation session per role.

7. Research Prototype

The presented approach proposes to enable stakeholders to validate behavioral specifications, i.e. story patterns, which are part of the collaborative scenarios the stakeholders are involved in (Chapter 4). These story patterns have to be executed, visualized, and animated in such a way that stakeholders can experience and judge them. Further, by enabling stakeholders to play in new story patterns (Chapter 5) or to explicitly describe continuations they expect based on experience (Chapter 6), requirements engineers can elicit the stakeholders' collaborative scenarios in individual stakeholder sessions. These concepts were implemented over the course of three years and together with a total of seven master's students who worked as student assistant or worked on the implementation as part of their respective master's theses [Kle11, Ric11, Teu11, Eic12]. The prototype was implemented in Eclipse[1] using EMF[2] and GMP.[3] To enable stakeholders to participate remotely, we developed a web-based user interface using Enterprise Java Beans which can be deployed on a JBoss[4] application server. The execution of story patterns is based on the *Story Diagram Interpreter* (SDI[5], cf. [GHS09]) which was developed by Stephan Hildebrandt.

The research prototype relies on an EMF Ecore model as domain model $\mathcal{D}$ (upper layer in Figure 7.1). Starting with s_{init}, states are dynamically instantiated during the simulation ($\mathcal{S}$, middle layer) based on this Ecore model (cf. *Dynamic EMF* in [SBPM09]). The story patterns $\mathcal{SP}$, in turn, are either derived from subsequent pairs of states (during *play-in*) or executed on these states (*play-out*). All story patterns conform to *their* metamodel that is defined as part of the SDI (illustrated abridged in Figure 7.2). Using its pre- and postcondition, a story pattern reflects the pair of states it was derived from. Each object contained in a story pattern is a `StoryPatternObject` representing an instance of a class defined in $\mathcal{D}$. The reference to the class in $\mathcal{D}$ which the object is an instance of is captured as the `classifier` of the `StoryPatternObject`. An association, on the other hand, is represented as a `StoryPatterLink` and refers to the domain model via its `eStructuralFeature` (cf. [GEHG13]).

[1] Version 3.7, `http://eclipse.org/` (accessed June 2013)
[2] Eclipse Modeling Framework Project, `http://eclipse.org/modeling/emf/` (accessed June 2013)
[3] Graphical Modeling Project, `http://www.eclipse.org/modeling/gmp/` (accessed June 2013)
[4] Version 7.0, `http://www.jboss.org/` (accessed June 2013)
[5] Available at `http://mdelab.de/update-site`, Version 2.3.3 (accessed June 2013)

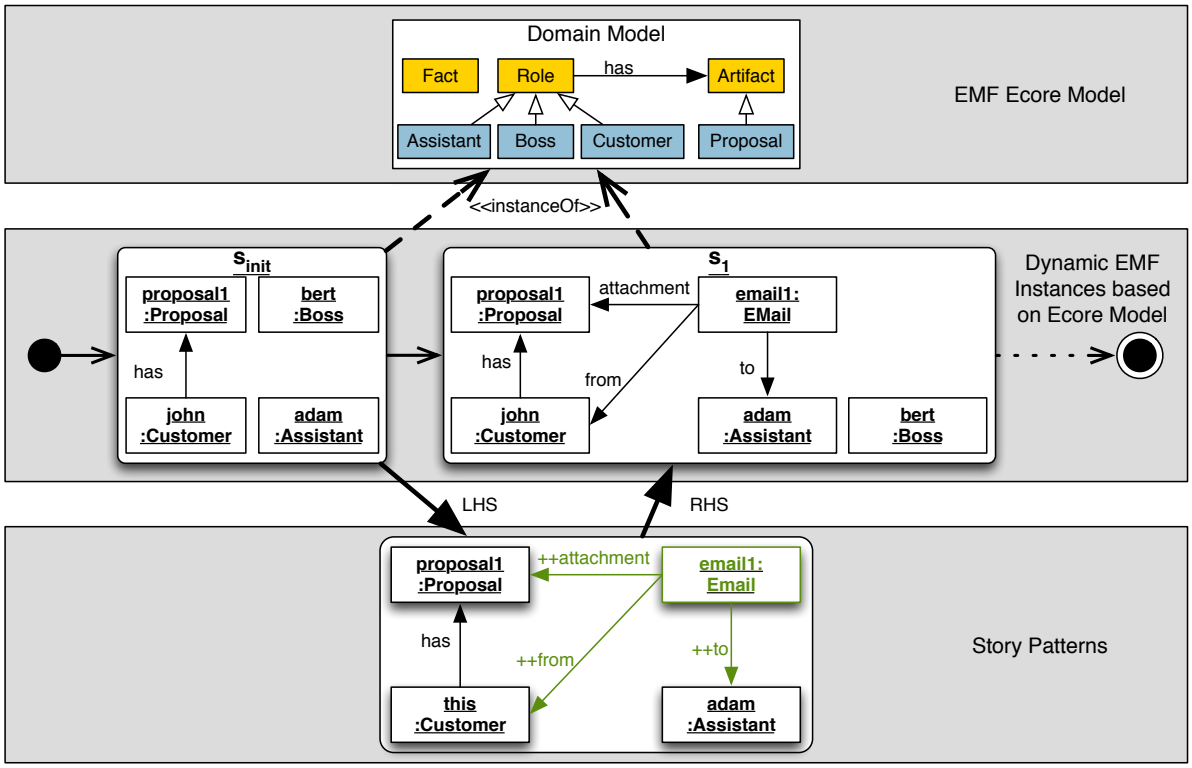


Figure 7.1.: In the research prototype, the domain model $\mathcal{D}$ is realized as an EMF Ecore model, the states $\mathcal{S}$ during the simulation are dynamic instances of this model, and story patterns $\mathcal{SP}$ are derived from these states (refinement of Figure 3.1d)

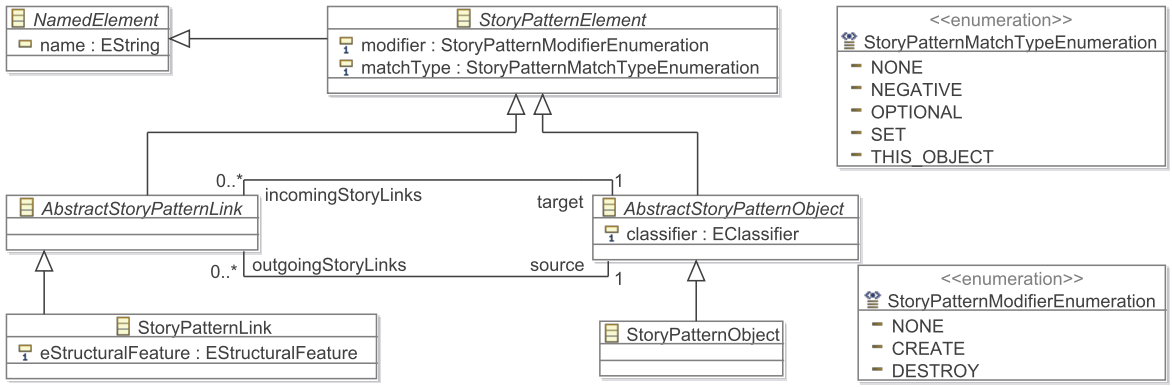


Figure 7.2.: Abridged metamodel of story patterns as defined by Hildebrandt's SDI (adapted from [GEHG13])

The files of a project that investigates collaborative scenarios consist of an Ecore file representing the domain model $\mathcal{D}$, a set $\mathcal{S}$ of states which are serialized as XMI[6] files, and a set $\mathcal{SP}$ of story patterns that reside in XML files as well.

[6] *XML Metadata Interchange* [OMG05c]

Section 7.1 describes the architecture of the research prototype – specific steps of its simulation loop (which was introduced and extended over chapters 4, 5, and 6) are illustrated using pseudo code listings in Section 7.2. Then, specific GUI interactions and the way they are translated into state changes are explained in Section 7.3. Further, Section 7.4 discusses the effort required to adapt the research prototype to a different domain. Afterwards, Section 7.5 summarizes this discussion of the research prototype.

7.1. Architecture

Figure 7.3 illustrates the deployment of our research prototype. Through a web browser, stakeholders interact remotely with the virtual prototype's front end. The mapping between the current state s_{cur} of the simulation and the visualization $s_{cur}|_r$ of a role r are realized by the `GameEngine`. Thus, the `GameEngine` is responsible of visualizing updates of $s_{cur}|_r$ in the participant's front end as well as modifying s_{cur} based on stakeholder actions observed in the front end.

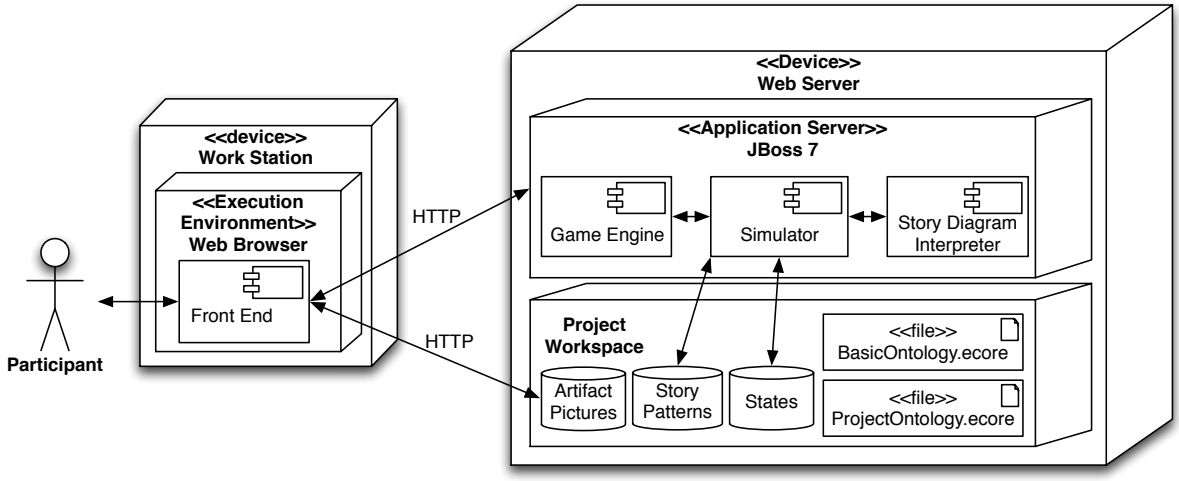


Figure 7.3.: UML deployment diagram of the research prototype

On the web server, the simulator runs on an application server and accesses the required models from within a project's workspace. For each workspace, there is a `ProjectOntology.ecore` which represents $\mathcal{D}_{specific}$ and, thus, extends the classes defined in the `BasicOntology.ecore` which represents $\mathcal{D}_{basic}$. Elements defined in either model are accessible via their corresponding namespaces.[7] All states in $\mathcal{S}$ are dynamic instances of the elements in these Ecore models. These states are serialized `.xmi` files which are stored by the simulator after each modification of the current state s_{cur} of the simulation. Further, all story patterns belonging to a project are saved as `.story` XMI files which

[7] Per default, these namespaces are `http://basicontology/` and `http://projectontology/`.

conform to the XML schema provided by the SDI (cf. Figure 7.2) and reference the namespaces of the Ecore models.

To enable stakeholders to intuitively recognize any of the artifacts they have to deal with, corresponding pictures can be saved on the web server. These can then be assigned to specific subclasses of `Artifact` so that each time a stakeholder wants to interact with an instance of an artifact, the assigned picture can be shown.

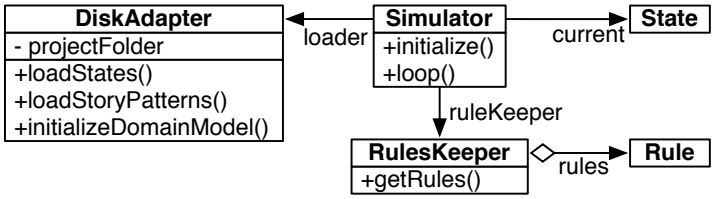


Figure 7.4.: The research prototype is initialized by loading the domain model, any scenario states, and the story patterns

To initialize a simulation session, the simulator employs the `DiskAdapter` which provides access to all files belonging to a project (Figure 7.4). Specifically, apart from initializing the EMF Ecore models, the `DiskAdapter` provides the initial state s_{init} which the simulator uses as the initial state of the scenario. The story patterns, on the other hand, are handed over to and managed by the `RulesKeeper`. A `Rule` contains a `StoryPattern` as defined by the SDI [GHS09] and a reference to a `Role` (or a subclass thereof defined in $\mathcal{D}_{specific}$) which the story pattern *belongs* to (cf. `thisObj` in Figure 7.5).

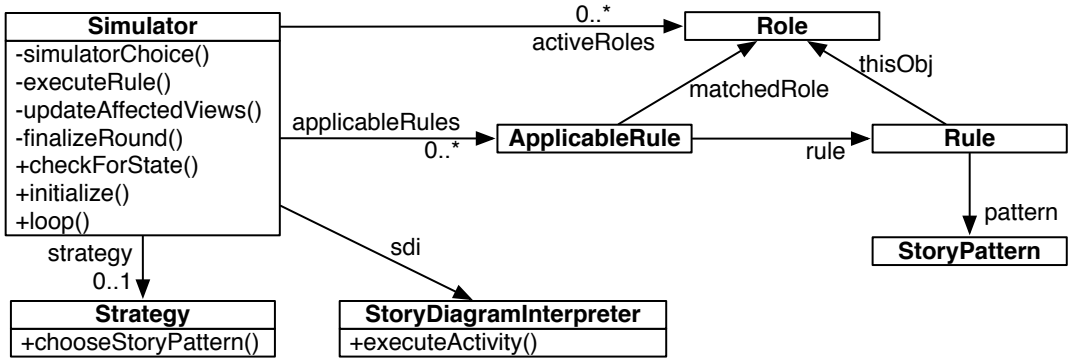


Figure 7.5.: For the play-out, the research prototype employs the SDI and strategies

During a running scenario simulation, the simulator can replay story patterns to simulate roles or derive new story patterns based on stakeholder interactions. To *play out* any story patterns, the simulator invokes the `StoryDiagramInterpreter` for each available `StoryPattern` to check whether a match for its preconditions can be found in the current state s_{cur} of the simulation. Consequently, the simulator obtains a set of `applicableRules` for which matches were identified (Figure 7.5). In case all story

patterns belong to specific roles such as `Assistant`, each applicable `Rule` is referenced by exactly one `ApplicableRule` and both point to the same specific role (via `thisObj` and `matchedRole`, respectively). If, however, a `StoryPattern` belongs to a generic role (e.g. `Role`) for which $\mathcal{D}$ defines subclasses (e.g. `Assistant`, `Boss`, and `Customer` as illustrated in Figure 7.1) the corresponding `Rule` might be applicable for more than one role. For instance, if a story pattern specifies, that `Role` may sign a `Proposal`, any subclass of `Role` (i.e. `Assistant`, `Boss`, and `Customer`) may provide a valid match. Consequently, in case `Assistant` and `Boss` both have access to a `Proposal` in s_{cur}, two instances of `ApplicableRule` would be created. While both instances would reference the same `Rule`, one points to `Assistant` and the other one to `Boss` as their respective `matchedRole`. This distinction allows the simulator to decide which role to simulate if a story pattern is executable for different roles.

If a `Strategy` was initialized for the simulation session, the simulator invokes it using its `applicableRules`. Then, the strategy chooses and returns one of those rules based on the strategy's goals (cf. Section 4.3). Otherwise, the simulator chooses at random which `ApplicableRule` the `StoryDiagramInterpreter` has to execute.

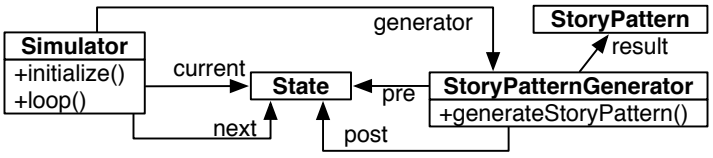


Figure 7.6.: Using the `StoryPatternGenerator`, the simulator can derive story patterns from states observed during the simulation

The *play-in* of story patterns depends on stakeholder modifications of the current state s_{cur}. These modifications are realized by the mapping between the domain-specific visualization and the underlying state in the `GameEngine` (cf. Figure 7.3). After a stakeholder participating as role r modifies s_{cur} and leads the simulation to s_{next}, the `StoryPatternGenerator` is automatically invoked to generate a `StoryPattern` based on the differences between $s_{cur}|_r$ and $s_{next}|_r$ (Figure 7.6).

7.2. Implementation of the Simulation Loop

This section explains how a simulation is run based on the simulation loop illustrated in Figure 7.7.

a) Load Initial Scenario State: To start a simulation session, the EMF Ecore files containing the domain model $\mathcal{D}$ have to be loaded first. Without these files, the references of the story patterns in $\mathcal{SP}$, which are loaded afterwards, would be invalid. Then, all states in $\mathcal{S}$, i.e. at least s_{init}, are loaded and initialized as *Dynamic EMF instances*.

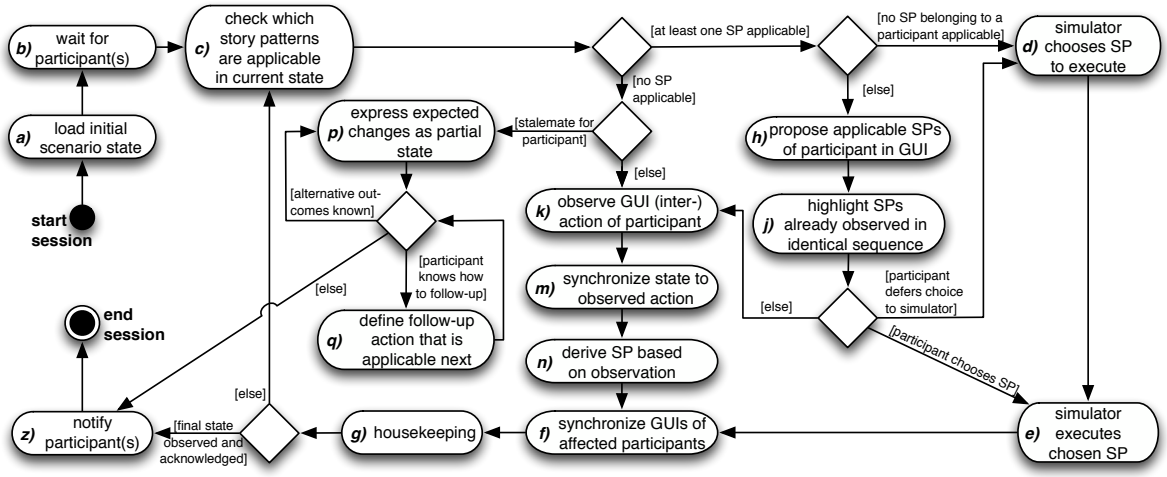


Figure 7.7.: Complete simulation loop including *Play-In, Play-Out,* and *Triggers*

As long as all files conform to the project's domain model captured in the EMF Ecore file, no errors occur. As per convention for the research prototype, each role that is defined in $\mathcal{D}$ must have exactly one instance in s_{init}.

b) Wait for Participants: After the session has been initiated, the prospective participants have to be invited. This can be achieved either by sending them a specific link[8] that can be generated by the prototype and which automatically logs the participant into a specific role, or by sending them a generic link to the web interface, i.e. without any parameters. All roles defined for the scenarios which are not yet chosen by other participants are offered in the web interface as illustrated in Figure 7.8.

After the invited stakeholders joined into the session, they have to explicitly confirm that they are ready by pressing a corresponding button labeled *"I'm ready"*. As soon as all participants who logged into the simulation are ready, the simulation starts in $s_{cur} := s_{init}$. Further, a list of the roles that are enacted by the participants can be accessed as `activeRoles`.

c) Check which Story Patterns are Applicable in Current State: To obtain the set $\mathcal{SP}_{applicable}$, the simulator has to check for matches for the left-hand side of each story pattern in $\mathcal{SP}$ in s_{cur}. To ensure that the current state s_{cur} itself does not change due to an unintended side-effect from any of the story patterns, test versions without any side-effects are created for these checks. Thus, for each story pattern $SP_i = (LHS_i, RHS_i)$, a test version $SP_i^{test} = (LHS_i, LHS_i)$ is created – a method described and employed in Gabrysiak's master's thesis [Gab09] for Fujaba [GZ04]. In our research prototype,

[8] `http://[your.domain]/webversion/?project=[projectTitle]&role=[nameOfInvitedRole]`

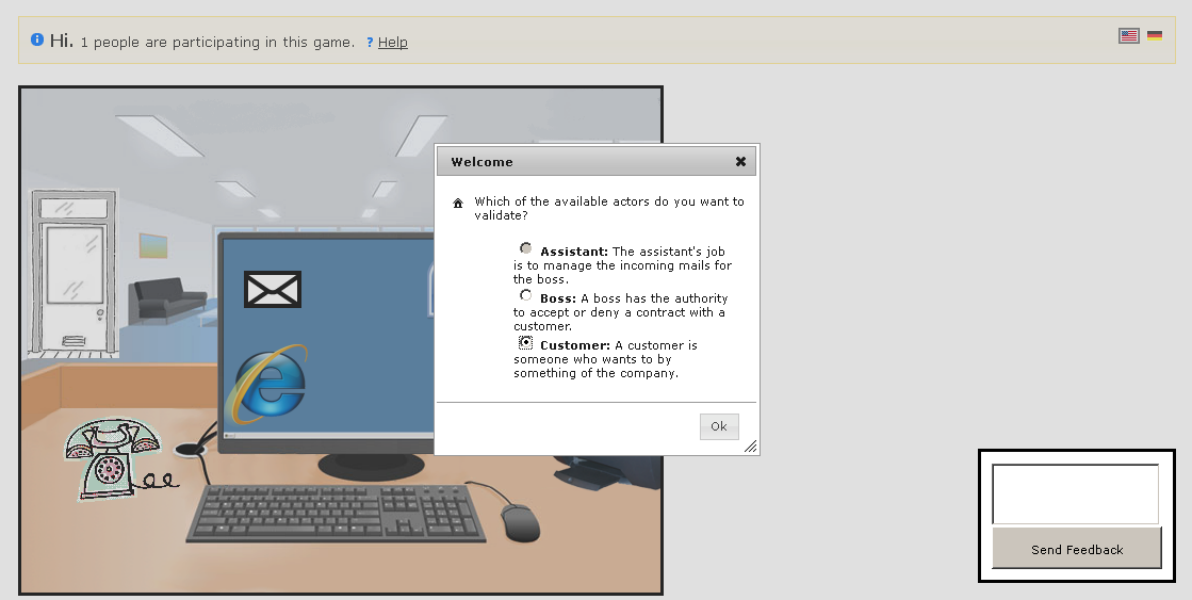


Figure 7.8.: All roles are offered to the participant, excluding the ones already chosen by other participants (`Assistant`, in this case)

this check is performed by the SDI which is invoked using the current state s_{cur} and an adapted version SP_i^{test} of the story pattern which is checked (cf. line 11 in Listing 7.1). Before this check is performed, the role the story pattern belongs to is chosen (`thisObj`). Then, all roles which are instances of `Role` and its subclasses are checked to identify those roles which can execute this behavior. Thus, only roles that can *match* the precondition of the story pattern are further tested by the SDI. If the SDI returns a suitable match, i.e. `sdi.executeActivity(`$\mathtt{SP_i^{test}}$`, r, `$\mathtt{s_{cur}}$`)`$\neq \emptyset$, the story pattern SP_i is applicable for the instance `r` of role. Multiple potential matches for one instance of a specific role are ignored since all objects are considered to be distinguishable (cf. Section 3.1.1). If the `this` object is a rather generic `Role`, all subclasses are considered suitable matches for the `this` object and, thus, may provide matches for the tested story pattern. For each of these matches, references to the applicable `Rule` and the matched instance of `Role` whose type is identical, or a subclass of, rule's `thisObj` are saved as an `ApplicableRule` (cf. lines 12–15 in Listing 7.1). All of these matches are then added to the set of applicable story patterns which is returned and handed over to the subsequent steps within the simulation loop.

If the simulator uses a strategy which requires a limited look ahead, the same method `checkForState()` is invoked for each of the states that needs to be explored until the maximum depth of the look ahead is arrived at.

h) Propose Applicable Story Patterns of Participants in GUI: If at least one of the applicable story patterns belongs to a participant, i.e. a role that is played by a

```
1  public ApplicableRule[] checkForState(Rule[] rules, State s_cur) {
2    applicableRules = new ApplicableRule[];  // reset Simulator's cache
3    for (Rule rule in rules){
4      StoryPattern sp = rule.getPattern();   // extract SP from rule
5      StoryPattern sp_test = new StoryPattern();
6      sp_test.LHS = sp.getLHS();  // same precondition as story pattern
7      sp_test.RHS = sp.getLHS();  // ensures that no side-effect is encoded
8      Role thisObj = rule.getThisObj();       // role the SP belongs to
9      for (Role r in s_cur.getRoles()) {      // find all instances of role
10       if (r.instanceOf(thisObj)) {
11         if (sdi.executeActivity(sp_test, r, s_cur) != null) {
12           Applicable appRule = new ApplicableRule();
13           appRule.matchedRole = r;        // role which can be simulated
14           appRule.rule = rule;            // rule of applicable SP
15           applicableRules.add(apprule); // save partial match
16         }
17       }
18     }
19   }
20   return applicableRules;
21 } // NOTE: s_cur did not change throughout this method
```

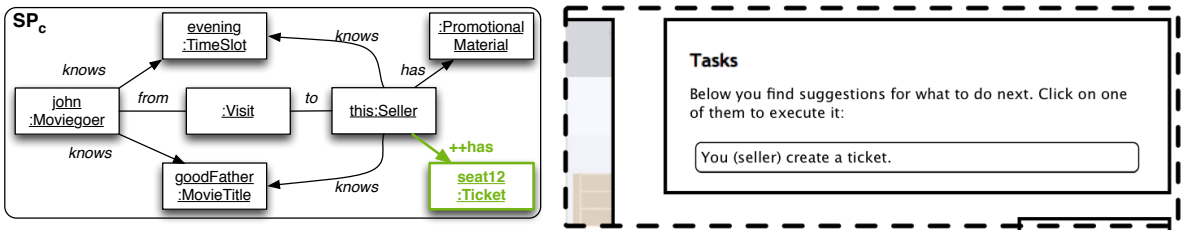
Listing 7.1: Searching for story patterns in $\mathcal{SP}$ which are applicable in s_{cur} (step c)

stakeholder (cf. lines 3–4 in Listing 7.2), this story pattern is considered as behavior that can be *expected* to occur. To allow participants to recognize and understand which options were already captured and, hence, which alternatives of the scenario were already covered in prior sessions, the story pattern is proposed to the corresponding participant (cf. lines 5–8 in Listing 7.2). Of course, presenting a formal story pattern to a stakeholder is not a viable option – a natural language representation, on the other hand, is easier to understand (cf. Figure 7.9). Such a representation can easily be generated – the more details are included in the domain model, the more understandable the resulting NL representation will be (cf. [GEHG13]). As discussed in Section 4.1, it suffices to present the changes encoded by the story pattern, since these changes allow the participant to recognize the activity and to judge whether it should be applicable based on the current state s_{cur} of the simulation. This is illustrated for SP_c in Figure 7.9. Furthermore, the proposed option contains JavaScript snippets which return an identifier for the story pattern as soon as the participant clicks on the proposed NL representation.

j) Highlight Story Patterns Already Observed in Identical Sequence: All observed traces are saved alongside the simulation. This enables the simulator to determine whether any of the prior simulation sessions covered an identical sequence of states and transitions between them. If any of the sequences reach up to s_{cur}, the applicable story

```
1  public void proposeOptions(ApplicableRule[] applicableRules) {
2    for (ApplicableRule appRule in applicableRules) {
3      Role role = appRule.getMatchedRole(); // SP executable by role
4      if (activeRoles.contains(role)) {     // is this role enacted?
5        Rule rule = appRule.getRule();      // retrieve rule
6        StoryPattern sp = rule.getPatter(); // retrieve applicable SP
7        String sp2nl = transformIntoNaturalLanguage(sp); // NL generation
8        proposeOptionToParticipant(sp, sp2nl, role);     // shows NL in GUI
9      }
10   }
11 }
```

Listing 7.2: Proposing natural language representations of changes encoded in applicable story patterns to participants (step h)



(a) Story pattern SP_c belongs to the role seller (b) Changes encoded in SP_c are presented to a seller

Figure 7.9.: When the story pattern SP_c is applicable for a stakeholder participating as a seller, it is proposed in the participant's GUI (cf. step h in Figure 7.7)

patterns may already have been observed. All story pattern for which this is true can then be highlighted accordingly.

d) Simulator Chooses Story Pattern to Execute: Based on the set of `applicableRules`, the simulator chooses one of them randomly for execution (cf. line 4 in Listing 7.3). Participants may also defer *their* choice of how to continue to the simulator instead.

If only one stakeholder participates, a default strategy prioritizing story patterns which *affect* this participant's role (i.e. change $s_{cur}|_r$, cf. discussion in Section 4.3.2) is used. Therefore, this strategy checks which of the applicable story patterns affects the participant through the creation, deletion, or modification of objects that are directly connected to the corresponding role. This also includes story patterns which either create, modify, or delete associations connected to this role. For instance, if a story pattern creates an email addressed to this role (i.e. linked to it via `to`), this story pattern would be prioritized over others without an impact on $s_{cur}|_r$.

e) Simulator Executes Chosen Story Pattern: The SDI is used to execute the chosen story pattern referenced by `chosenAppRule`. This, in turn, simulates the corresponding activity for the role that the `chosenAppRule` points to as `matchedRole`. Listing 7.4 illustrates how the simulator reaches the follow-up state s_{next}.

k) Observe GUI (Inter-) Action of Participant: As described in Section 3.2, the graphical user interface provided to participants has interactive elements which enable participants to change the current state of the simulation. Further, these interactive elements encapsulate some of the concepts in $\mathcal{D}$. Thus, the participants can submit the data necessary to instantiate those concepts using the corresponding interactive parts of the GUI. For example, if an email is composed, its subject, its content, and IDs of any of the participant's artifacts which were attached are entered by the participant to be sent to the server along with a list of roles who are supposed to receive a copy of this email (cf. Figure 7.10 and Table 7.1). Specifically, clicking on the email icon and choosing to write a new email leads to the corresponding window. Other roles which are defined in $\mathcal{D}$ are enumerated as possible receivers, while the subject line and the body of the email are simple text fields. To attach an artifact, on the other hand, the participant can *drag and drop* it from his *Documents* field on the right onto the email window. Then, the email can be sent by pressing the button labeled *Send*. Of course, the email is only sent within the simulation.

m) Synchronize State to Observed Action: Based on the data collected from the participant's interaction with the interactive GUI elements, the simulator running on the server has to update the current state of the simulation. The interactive elements use JavaScript to send corresponding requests to the server informing it of any changes the participant enacted in the GUI. After the server has been informed, these actions are translated into model updates for the current state of the simulation based on the mapping between the GUI and the domain model. For instance, the data provided by a participant in order to write an email (illustrated in Figure 7.10) is used to instantiate

```
1 private ApplicableRule simulatorChoice(ApplicableRule[] applicableRules,
    Strategy strategy) {
2   ApplicableRule chosenRule = null;
3   if (strategy != null) {   // use strategy to choose what to do
4     chosenRule = strategy.chooseStoryPattern(applicableRules);
5   } else {                  // otherwise, choose randomly
6     chosenRule = applicableRules[random(applicableRules.size)];
7   }
8   return chosenRule;        // which SP to execute along with the role
9 }
```

Listing 7.3: The simulator chooses a story pattern randomly or using a strategy (step *d*)

```
1 private State executeRule(ApplicableRule chosenAppRule, State s_cur) {
2   Rule rule = chosenAppRule.getRule();
3   StoryPattern chosenSP = rule.getPattern();
4   Role executingRole = chosenAppRule.getMatchedRole(); // simulated role
5   State s_next = sdi.executeActivity(chosenSP, executingRole, s_cur);
6   return s_next;  // follow-up state
7 }
```

Listing 7.4: The SDI progresses the simulator into a new follow-up state (step e)

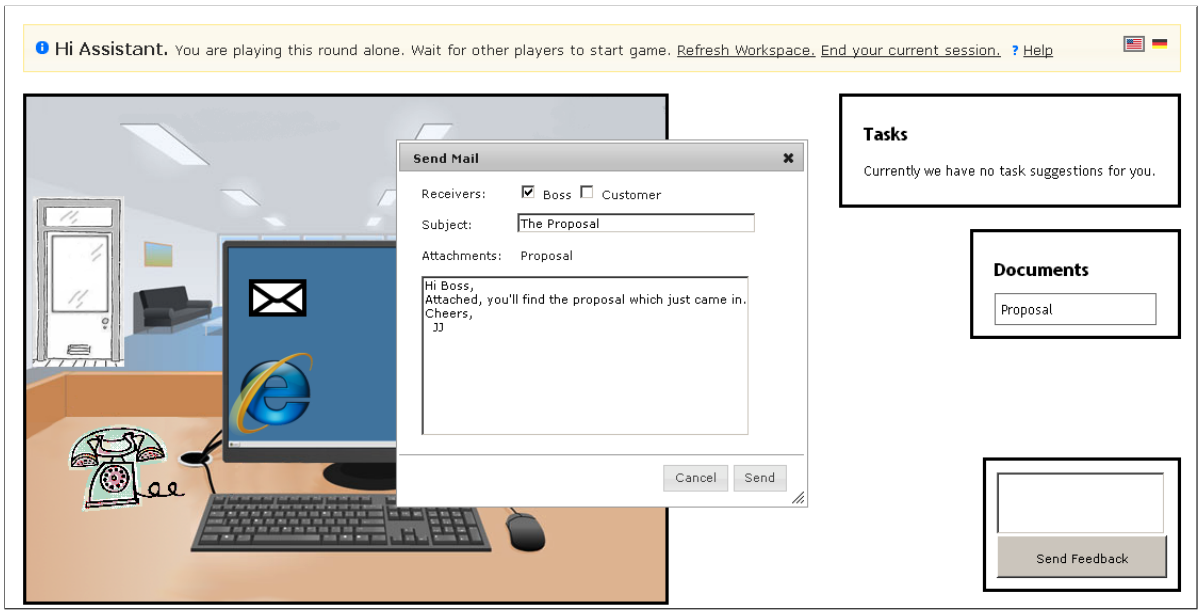


Figure 7.10.: As an `Assistant`, the participant writes an email informing the `Boss` about a `Proposal` that just arrived

an email object whose body and subject line are the ones provided by the participant. Further, the chosen artifacts are attached to the email and one instance of this email is created for *each* of the receivers.[9] Finally, each created email object is linked to the initiating participant (`from`) and to the individual receiving role (`to`).

More examples of how GUI interactions are translated into state updates are illustrated in Table 7.1. The mapping between concepts in $\mathcal{D}$, their representation in the participants' GUI, and how these concepts are instantiated based on inputs within the front end have to be created manually. However, this is only necessary once, since such a mapping can be reused for other projects covering the same domain.

At the end of this step, the simulation progresses to the follow-up state s_{next}.

[9] In the research prototype, this email duplication is used to keep track of who already read the email and who did not.

Participant's Action in Front End	Data sent from Client to Server	Modification of s_{cur} due to Participant's Action
Write email	In an email editor, the participant enters subject line and text (both `strings`), chooses receivers and may attach artifacts	A corresponding email object is created for each receiver
Read email	The participant selects an email from a list of all emails	The selected `Email` instance is consumed (deleted from the state)
Create an artifact	The participant selects an artifact to create from a list of all artifacts defined in $\mathcal{D}$	A corresponding instance is created and assigned to the participant's role
Process an artifact and name its state	Lines drawn on the artifact's picture and name of its new state	New `ArtifactState` is created and overlay is assigned to it
Visit another role	Target role the participant wants to visit	A `Visit` is instantiated between initiating and target role

Table 7.1.: Examples of GUI interactions (*left*) that enable participants to enter data (*middle*) which is sent to the server to update s_{cur} correspondingly (*right*)

n) Derive Story Pattern based on Observation: Based on the prior state (still referred to as s_{cur}) and the follow-up state s_{next}, a story pattern $SP_{derived} = (s_{cur}|_r, s_{next}|_r)$ can be generated in accordance to the visibility of the participant's role r.

Conceptually, the generation of the story pattern is quite easy as illustrated in Listing 7.5. However, this ignores the story pattern encoding employed by the SDI [DHP+12]. In this encoding, a story pattern consists of only one graph. Elements in this graph possess an attribute called `modifier` which indicates whether this element is *created* (all elements $\in RHS \setminus LHS$), *destroyed* (all elements $\in LHS \setminus RHS$), or remains without change (all elements $\in LHS \cap RHS$). These modifiers represent the changes which story patterns encode in their LHS and their RHS.

The actually implemented story pattern generation works as follows: in a story pattern, all objects and the links between them are represented using `StoryPatternObject`s and `StoryPatternLink`s, respectively (cf. Figure 7.2). After a `StoryPatternObject` is instantiated for each object $obj \in s_{cur}|_r \cup s_{next}|_r$, its `classifier` is set to the same type in $\mathcal{D}$ (i.e. class in the Ecore model) that the original object *obj* referred to. Afterwards, the same is done for all links in $s_{cur}|_r \cup s_{next}|_r$: `StoryPatternLink`s are created and associated with the sources and targets corresponding to their originals in $s_{cur}|_r \cup s_{next}|_r$. Instances of `StoryPatternLink` reference the links in $\mathcal{D}$ via their attribute `eStructuralFeature`.

Since states of the simulation are saved each time they are changed, they are available as XMI files which conform to $\mathcal{D}$ and can be compared using the Eclipse project

```
1 private StoryPattern generateStoryPattern(State lhs, State rhs, Role r) {
2   StoryPattern genSP = new StoryPattern();
3   genSP.LHS = reduceStateToVisibilityOfRole(lhs, r); // build lhs|_r
4   genSP.RHS = reduceStateToVisibilityOfRole(rhs, r); // build rhs|_r
5   return genSP;
6 }
```

Listing 7.5: Conceptually, the generation of a story pattern is quite simple (step n)

EMF Compare.[10] At this point, EMF Compare is used to create a *diff model* from the subsequent states, reduced to the visibility of the participant's role r. This diff model contains information on which elements from the prior state s_{cur} were removed to get to the follow-up state s_{next} and which elements were added. Using this information, the objects and links in the story pattern can be attributed accordingly. The `modifier` of each element which the diff model considers deleted is set to `DESTROY`, while elements which were added are set to `CREATE`. All other elements have a default modifier of `NONE`, indicating that they remain the same.

p) Define Trigger as Expected Continuation: A stakeholder participating as $Role_T$ has to be able to define her expectations ($s'_{sm}|_{Role_T}$) based on her perspective ($s_{sm}|_{Role_T}$) of the stalemate state s_{sm}. By modifying elements of this perspective, the expected changes can be defined. This capability was implemented in Eichler's NL4SP approach [Eic12]. His implementation generates a NL representation of a story pattern and allows stakeholders to consistently modify the underlying story pattern by directly interacting with the NL representation of these partial states (cf. Section 2.5). Thus, by removing, adding, or modifying elements in the NL representation of $s_{sm}|_{Role_T}$, stakeholders are empowered to express how they expect this specific state to change (resulting in $s'_{sm}|_{Role_T}$).

Since the definition of the expected changes between $s_{sm}|_{Role_T}$ and $s'_{sm}|_{Role_T}$ is covered, the remaining parts of a trigger $t = (s_{sm}, Role_T, Role_{Req}, Role_{Resp}, s'_{sm}|_{Role_T})$ have to be defined. Essentially, the participant (who is participating as $Role_T$) only needs to pick different roles out of the ones defined in $\mathcal{D}$ to complete the trigger. As for the research prototype, the most complex part of enabling participants to define triggers (i.e. the description of the expected partial state) is covered by Eichler's NL4SP implementation.

q) Define Follow-Up Action: Similar to how stakeholders can use the prototype discussed in step p to define changes, they can also define *follow-up actions* (cf. Figure 7.11). A follow-up action is considered equivalent to a story pattern (cf. Section 6.1.2). Thus, provided a state $s_F|_{Role_T}$ exists, a story pattern $SP_F = (s_F|_{Role_T}, s_F|_{Role_T})$ can be built and presented in NL. This story pattern does not contain any changes, since its

[10] `http://eclipse.org/emf/compare/` (accessed June 2013)

LHS and its RHS are identical. Consequently, the stakeholder can then use the vocabulary defined by $\mathcal{D}$ to express how he or she would continue. Thereby, the RHS of SP_F is modified, leading to a follow-up action $SP'_F = (s_F|_{Role_T}, s'_F|_{Role_T})$ which encodes how the stakeholder would follow up upon arriving in a state $s_{cur} \supseteq s_F|_{Role_T}$. Stakeholders can use the NL4SP prototype for this task as well.

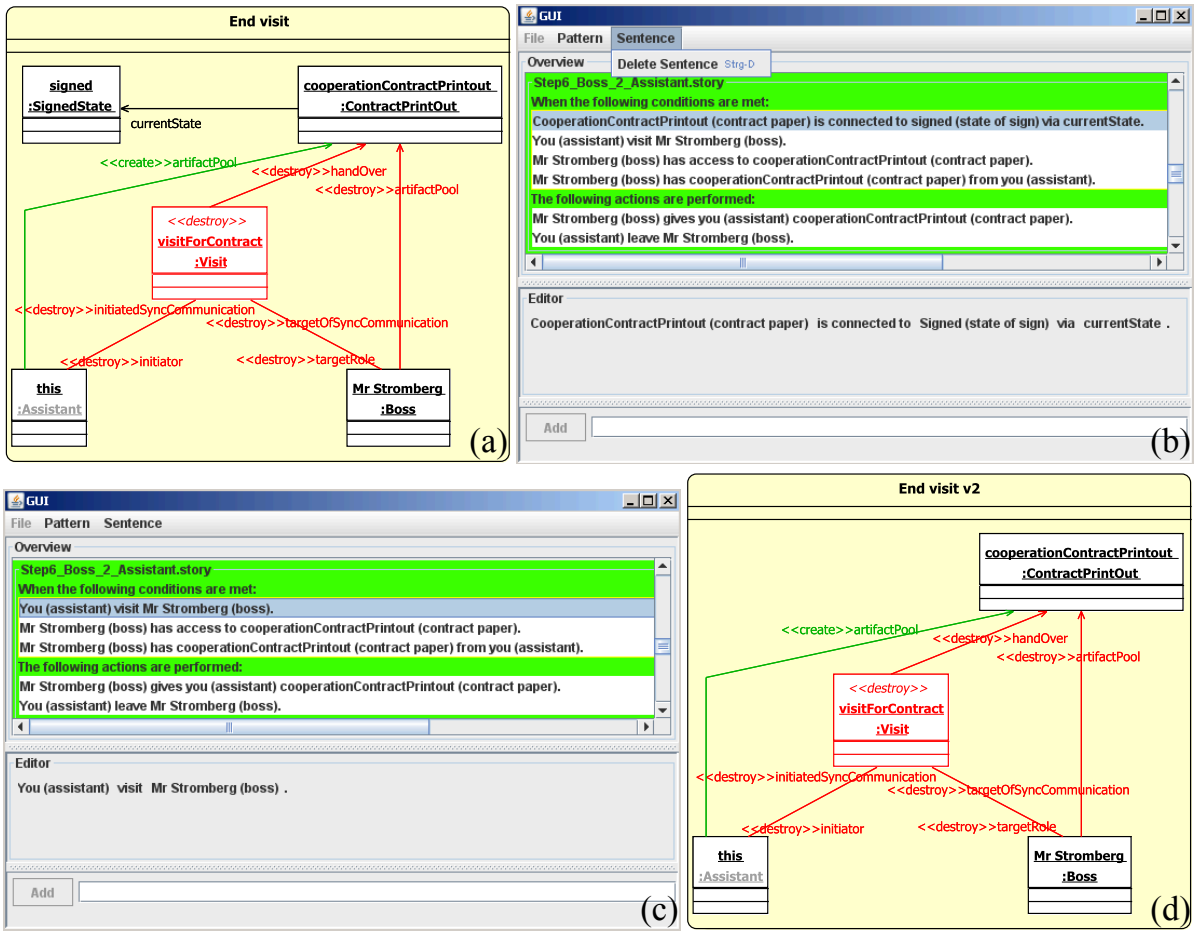


Figure 7.11.: Stakeholders can modify a story pattern (a and d, before and after, resp.) through a NL interface (b,c) to, e.g., remove the incorrect assumption that only `signed` contracts are handed over (adapted from [GEHG13])

f) Synchronize GUIs of Affected Participants: Any transition between s_{cur} and its follow-up state s_{next} may affect each of the roles. Thus, each stakeholder participating as a role that is *affected* by such a transition requires updates for his or her corresponding graphical user interface (cf. Listing 7.6). Depending on how the context of a role ($s_{cur}|_r$) changed, the participant's GUI is updated accordingly based on the mapping illustrated

in Table 7.2. Since the server does not push changes to the GUIs which have to be updated, each client sends a pull request every two seconds.

```
private void updateAffectedViews(State s_cur, State s_next) {
  for (Role r in activeRoles) {
    State s_cur_r  = reduceStateToVisibilityOfRole(s_cur, r);
    State s_next_r = reduceStateToVisibilityOfRole(s_next, r);
    if (s_cur_r != s_next_r) {       // then: role affected by transition
      updateViewOfRole(r, s_next_r); // update affected perspective
    }
  }
}
```

Listing 7.6: After establishing which roles that are currently played are affected, their individual perspectives are updated to reflect the new state (step *f*)

g) Housekeeping: Before the simulation loop starts over by entering step *c* again, minor adjustments and checks are necessary. First of all, the follow-up state s_{next} is set as the current state for the next round: $s_{cur} := s_{next}$. Then, all terminal states $\in S_{term}$ are checked, whether one of them are a match for the current state of the simulation and, thus, has been reached. For this check, a story pattern is created to employ the SDI. Since no changes are performed by this story pattern, no `this` role needs to be identified beforehand. The simulation is ended if all participants acknowledge this terminal state.

```
private Bool finalizeRound(State[] terminals, State s_cur, State s_next) {
  s_cur = s_next;                // follow-up state is now current one
  for (State s_term in terminals) {
    StoryPattern sp_test = new StoryPattern();
    sp_test.LHS = s_term;
    sp_test.RHS = s_term;
    if (sdi.executeActivity(sp_test, null, s_cur) != null) {
      return askParticipantsAcknowledgement(s_term);
    }
  }
  return false; // loop continues if no match is found
}
```

Listing 7.7: If a final state can be matched, participants are asked to confirm that they consider the scenario to be over – if not, the simulation continues

z) Notify Participant(s): After a final state has been matched and acknowledged, the simulation session is terminated and each participant is asked to provide additional feedback concerning the scenario and the role that was just used to play through this scenario (cf. Figure 7.12).

Modification of $s_{cur}\|_r$	Data sent from Server to Client	Update in Participant's GUI
Role received an `Email`	`Email` instance with all values (such as body and subject line) necessary to present this email to the participant	Email icon starts to flash (until participant clicks on it to see the email)
Role receives a `Call`	`Call` object, including all values (such as initiating role linked via `from`) necessary to indicate the call to the participant	Telephone icon is animated, ringing sound is played, and name of the calling role starts to flash above the icon (until participant clicks on it to start a conversation via asynchronous chat)
Role is targeted by a `Visit`	`Visit` object including all values (such as initiating role linked via `from`) necessary to indicate the visit to the participant	Door icon is replaced by an open door and a window is shown to inform the participant that (s)he is visited by the initiating role (this window also allows the roles to communicate via an asynchronous chat and exchange `AnalogueArtifacts`)
Role receives a specific `Artifact`	Instance of the `Artifact`, including a link to its picture on the server and, if applicable, its `ArtifactState` and associated pictures	Artifact is presented in the box labeled *Documents* (clicking on it loads and presents the artifact's picture in a new window)

Table 7.2.: Examples of state modifications (*left*), the data that is sent to the clients of the affected participants, and its impact on the participant's GUI (*right*)

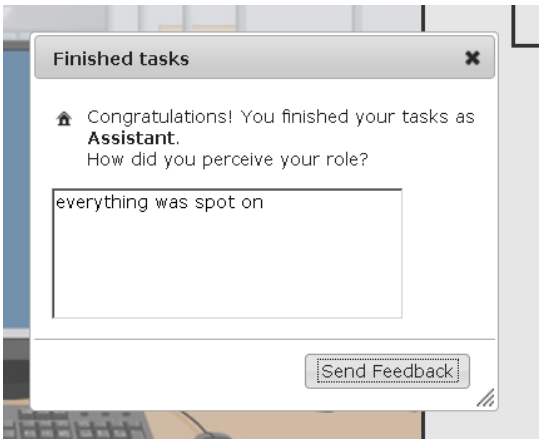
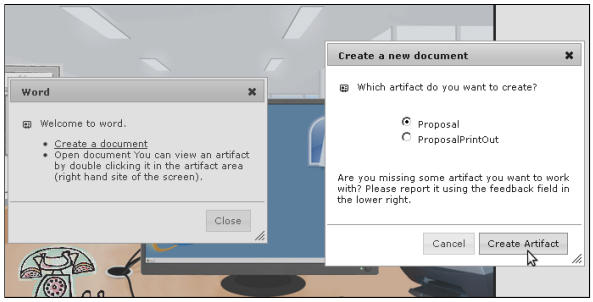


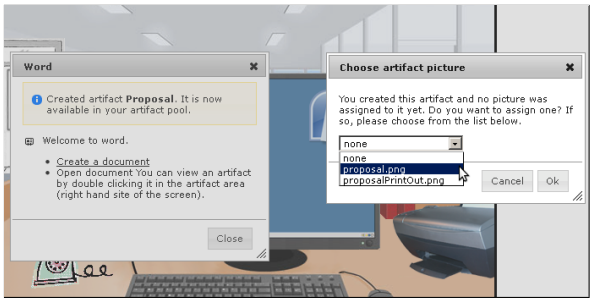
Figure 7.12.: When a session is terminated, participants are asked for additional feedback

7.3. Interacting with Artifacts

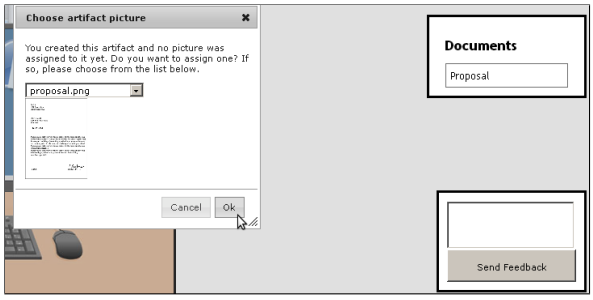
Apart from interacting with other roles, the creation and modification of artifacts are essential activities within collaborative scenarios. To create an artifact, participants may use the icon of a well-known *word processor*. By clicking on the corresponding icon, a list of all subclasses of `DigitalArtifact` and `AnalogArtifact` defined in $\mathcal{D}$ are presented to the participant (Figure 7.13a). Upon clicking on one of these classes, a corresponding instance is initialized and assigned to the role of the participant (Figure 7.13d). To be recognizable for the stakeholders, each artifact may have a picture assigned to it (cf. Figure 3.3). If the artifact's class has no assigned picture yet, all pictures in the project's `artifact/` folder are presented for the participant to choose from (Figures 7.13b and 7.13c). The chosen picture is then assigned to the artifact instance which can then be recognized more easily by other stakeholders.



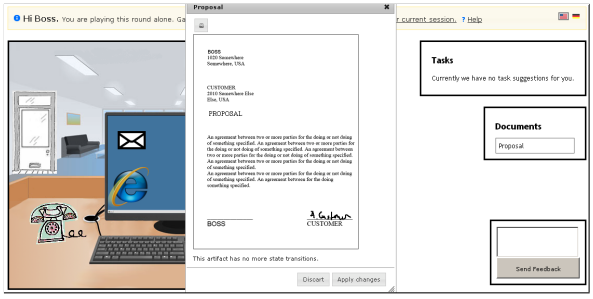
(a) A participant picks an artifacts from the ones defined in $\mathcal{D}$



(b) If no picture was assigned, the participant picks one from the `artifacts/` folder



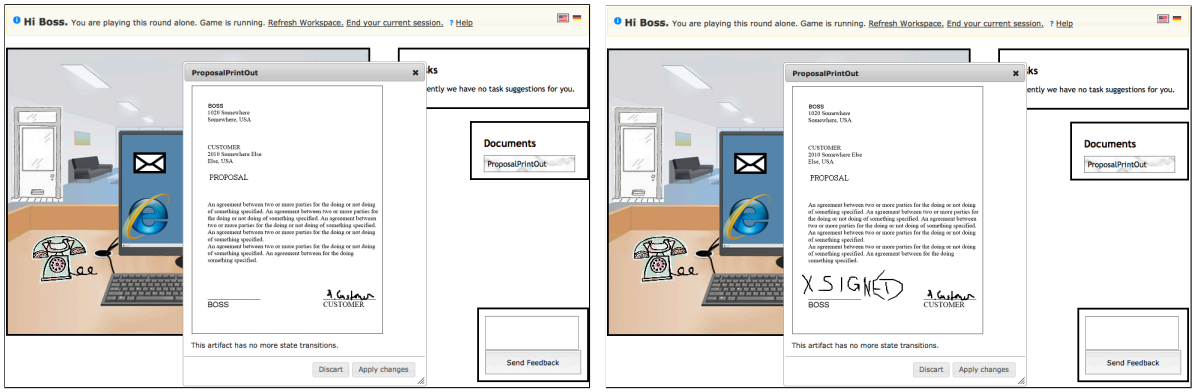
(c) A preview of the chosen picture is shown



(d) Then, participants interact with the artifact

Figure 7.13.: A participant may create any of the artifacts defined in $\mathcal{D}$ which creates an instance that the participant's role has access to

Furthermore, participants need to be able to modify artifacts in order to, e.g., indicate processing steps associated with the artifact. All artifacts that a participant's role has access to are enumerated in the GUI within the *Documents* list (illustrated in Figures 7.10 and 7.14). Clicking twice on one of the artifacts shown in this list opens a new view presenting the picture of the artifact to the participant. From this view, participants



(a) Before and ... (b) ... after a `Boss` signs a `ProposalPrintout`

Figure 7.14.: Using a mouse, participants can draw on artifacts they have access to

may draw on top of this picture using their mouse. This enables them to express more complex activities such as *signing* or *stamping* documents, or even annotating these artifacts using free text which can be drawn into the picture as illustrated in Figure 7.14b. This feature was implemented using HTML5 canvas.[11] All annotations, i.e. everything that was drawn into the overlay, are saved using JSON.[12]

In the current version of the research prototype, an overlay that is created for an artifact instance remains assigned to it and active for the remainder of this session, apart from being saved in the same folder as the artifact pictures. Further, the participant may choose which `ArtifactState` the artifact instance is in. If no `ArtifactState` has been defined, the participant is able to create a new one to describe the artifact modifications. This information is potentially available for later sessions to replay activities and how they have manipulated the affected artifacts.

7.4. Adaptability

Although the current implementation already covers numerous types of collaborative scenarios which may occur in office environments, the research prototype may still need to be adapted to cover, e.g. unexpected types of communication in new domains. The simulation loop is domain-agnostic, since it only relies on generic concepts such as `Role` and `Artifact` in $\mathcal{D}_{basic}$. Therefore, as long as $\mathcal{D}_{basic}$ remains the same, the simulation works for all states and story patterns which can be defined based on $\mathcal{D}$.

The domain-specific GUI, however, needs to be adapted for each domain. In the research prototype, the office background and the employed visual metaphors represented

[11] `http://www.w3.org/TR/2009/WD-html5-20090825/the-canvas-element.html` (accessed June 2013)

[12] JavaScript Object Notation, cf. [Cro06] and `http://json.org/` (accessed June 2013)

by different icons are identical for all stakeholder groups. Thus, the research prototype does not support scenarios in which stakeholder groups require different workplace visualizations. Further, since the mapping between the GUI and the simulation is hard coded, the mappings in Tables 7.1 and 7.2 have to be adjusted accordingly to reflect how options offered in the GUI modify s_{cur} and how changes of s_{cur} have to be visualized in the GUI. Specifically, this applies if new modes of communication are introduced in $\mathcal{D}$. While the domain-agnostic simulation loop is not affected by the introduction of, e.g., tube mail, a suitable visual metaphor needs to be found and defined for both mappings.

7.5. Chapter Summary

Over a period of three years and together with seven student assistants, the concepts described in this thesis have been implemented in Eclipse and EMF. In the resulting research prototype, the requirements engineer can initialize a new project by defining roles and an initial state for the collaborative scenarios to be elicited. Then, the research prototype can already be used for remote simulation session for stakeholders.

Furthermore, this prototype has been improved, extended, and evaluated in multiple master's theses [Eic12, Kle11, Ric11, Teu11]. The next chapter discusses the experiments in which the research prototype was used to evaluate our concepts.

8. Evaluation

To evaluate the goals discussed within this dissertation, we conducted different experiments using our research prototype. Most importantly, we investigated whether our virtual prototypes fulfill the requirements of a prototype which can be used for requirements engineering. Thus, these virtual prototypes need to be *understandable for stakeholders* ($\mathbf{TG}_{2A}$) – otherwise, they would only be an alternative representation of the models which might be just too time consuming to explain to be useful. Another important strength of physical prototypes is the fact they can be produced inexpensively which allows the designers or engineers to *iterate their ideas quickly* ($\mathbf{TG}_{2C}$). Again, this is only the case, if the result is intuitively understandable for the stakeholders. Still, from an requirements engineer's perspective, the generation of a virtual prototype has to be quick and include as much automation as possible – otherwise, the costs of deploying these prototypes would be too high to be feasible. Furthermore, physical prototypes are usually *easy to modify* ($\mathbf{TG}_{2B}$) or reshape, depending on their materials. Thus, empowering stakeholders to actively change a virtual prototype by *complementing collaborative scenarios* ($\mathbf{TG}_{1B}$) might lead to a similar engagement, especially if the prototype affords *immediate feedback* regarding these changes ($\mathbf{TG}_{1A}$). Consequently, in this chapter, we evaluate whether our model-based virtual prototyping approach fulfills these properties which a prototype requires to be useful and feasible.

Section 8.1 starts by explaining *t-tests* and *threats to validity* which apply uniformly for all the experiments we conducted. Most importantly, stakeholders have to be able to intuitively understand the virtual prototypes, which our research prototype derives. This was evaluated empirically in two experiments covering $\mathbf{TG}_{2A}$ which are discussed in Section 8.2. Specifically, these experiments answer the question whether stakeholders can interact using our research prototype (step k in Figure 8.1) and stakeholders correctly interpret changes which are visualized in the GUI (step f). The requirements engineer's ability to feasibly iterate the story patterns reflecting the stakeholders' scenarios depends on the time required to set up a virtual prototype as well as the overall degree of automation of our approach. Thus, Section 8.3 presents measurements of the time it takes to set up a virtual prototype (step a) and describes the overall degree of automation of the research prototype's simulation loop (discussion of all steps in Figure 8.1), which covers $\mathbf{TG}_{2C}$.

Section 8.4 presents an evaluation of the stakeholders' ability to modify the models belonging to the virtual prototype which is partially based on our publication [GEHG13] and covers $\mathbf{TG}_{2B}$. This experiment evaluates the natural language generation and the corresponding interface which partially cover steps p and q. Additionally, this section

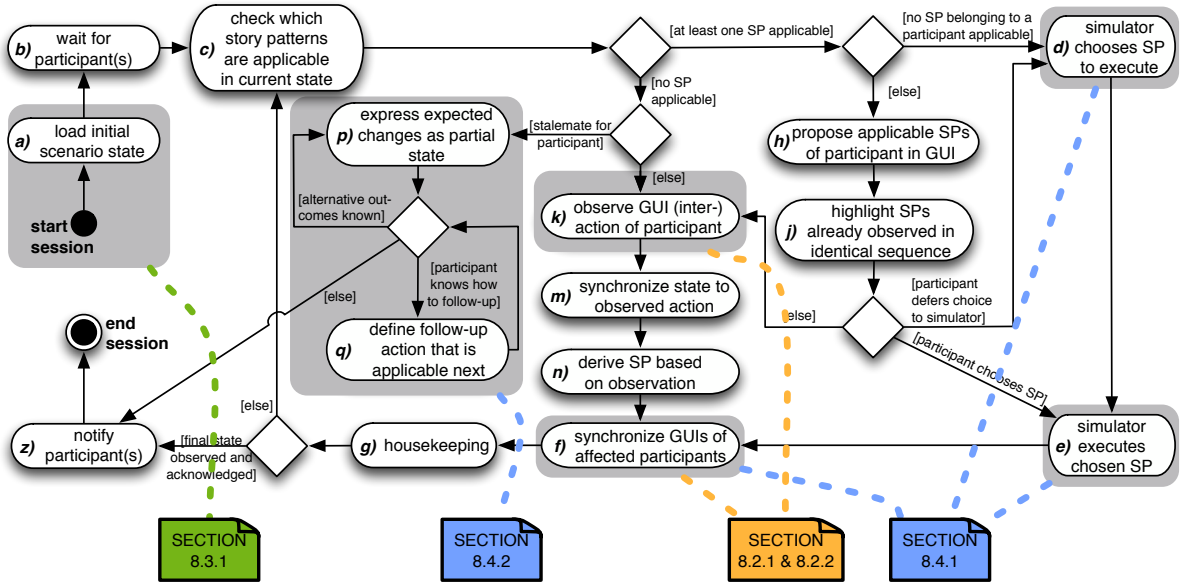


Figure 8.1.: The highlighted steps of the simulation loop are evaluated in the assigned sections

discusses how stakeholders may modify the prototype itself through their interactions to complement scenarios started by other stakeholders (covering $\mathbf{TG}_{1B}$). Furthermore, it is discussed how such additions lead to immediate feedback for the participating stakeholders (covering $\mathbf{TG}_{1A}$). This discussion partially covers steps *d*, *e*, and *f* as illustrated in Figure 8.1. Section 8.5 summarizes the results of our evaluation.

8.1. Preliminaries

As part of our evaluation, we conducted different experiments either with multiple groups of software engineering students or with up to two groups of students from unrelated disciplines, i.e. without a technical background. This section explains the common threats to validity which can be excluded for all the experiments described in this chapter. Additionally, t-tests are explained, which are used throughout this chapter to evaluate the statistical significance of our results.

Threats to Validity (based on [WRH+00, MDF05]): All of the experiments we conducted took place at only one point in time for each participant or group of participants. Thus, our experiment designs already addressed specific threats to the internal validity of our experiments. During the time frame in which we conducted these experiments, there

was no event of such magnitude that it might have had an impact on the participants and their performance, i.e. there was no threat due to the *history*.

In most of our experiments, the duration was less than one hour. In this short time frame, no biological or psychological changes affecting the subjects' ability to participate or their overall performance are to be expected. Only one experiment took place for a total of up to four hours (cf. Section 8.2.1). However, even in this setting, the participants were explicitly allowed to take breaks or eat as they saw fit. Consequently, the threat of biological and psychological changes, i.e. *maturation*, can be excluded. Moreover, due to the short overall duration of the experiments, there was no drop out or *mortality* of the participants, which allows us to exclude this threat. For the duration of each experiment series, the instruments used to measure the dependent variables remained the same. Thus, we can exclude the threat of *instrument change*.

Each of our experiments was only conducted once with each participant. Therefore, we can also exclude the threat of subjects performing better due a learning effect or *testing effect*. Since none of the experiments discussed in the evaluation rely on pre- and posttests, we can also exclude the effect of the *regression toward the mean*, which, otherwise, might influence our results.

T-Test: As suggested by Zikmund et al. [ZBCG10], we applied the *independent samples t-test* to test the differences between the means taken from two independent groups. Furthermore, they point out that this test is suitable as long as the variance of both groups can be assumed to be "approximately equal". The independent samples t-test was used throughout this chapter. Its results are provided in the form of p values which indicate the statistical significance of the difference between the means of two groups [MDF05]. The typical convention, as argued by Marczyk et al. [MDF05], is a five percent level for statistical significance (cf. [Sar05, ZBCG10]). Thus, if a t-test yields a p value of 0.05 or less, the difference between the means of the two groups is considered to be significant and not a result of mere chance. This, in turn, implies that the treatment of the experiment group resulted in a significantly different outcome which allows us to determine whether the treatment had the hypothesized effect. Throughout this chapter, all figures illustrating experimental results also contain *error bars* presenting the corresponding 95% confidence interval.

8.2. Understandable Representation

Only if they can understand a virtual prototype and the concepts it manifests, stakeholders are able to explore and complement scenarios involving them. In order to evaluate the understandability and, hence, whether stakeholders are able to interact with our research prototype, we investigated how a stakeholder and four students gathering requirements from him interact using the research prototype (Section 8.2.1, partially based on [GGS12]). Additionally, in order to ensure that the chosen metaphors in the research

prototype's web-based interface are suitable, we asked students to interpret interactions with the user interface that were captured on video (Section 8.2.2, partially based on the GUI evaluation in [GGB12]).

8.2.1. Interacting to Validate Scenarios

To investigate how students elicit, model, and validate scenarios involving multiple stakeholders, we had nine groups of four students each interview a stakeholder, model their insights, and then validate what they learned in a final interaction with the stakeholder. Three of these groups used the research prototype without its simulation capabilities. Therefore, they solely relied on its interaction capabilities. While three other groups were allowed to flexibly document their findings however they saw fit using pens and paper, the remaining three groups used a UML modeling tool. Thus, our research prototype was compared against a formal modeling approach leading to UML models and an informal approach which led to more understandable requirements to be validated.

Setup: As part of an undergraduate modeling lecture, a total of 36 students from our institute used three different specification methods. As participants of the lecture, the students had already formed teams of four students each. In nine groups of four students each, they elicited requirements (up to 55 min), modeled their findings (up to 110 min), and validated their result (up to 50 min). For the elicitation and validation, each group talked to one out of nine stakeholders who was randomly assigned to the group. The students talked to their distinct stakeholder about a speculative online supermarket and how the stakeholders, in his or her role as a supporting call center agent, solved problems related to, e.g., belated deliveries.

All nine stakeholders were simulated by students without a software engineering background. None of them were familiar with the scenario – however, all of them were uniformly instructed about possible scenarios for two hours prior to the experiment. This ensured that all students interacting with these stakeholders had the opportunity to gather equivalent scenarios.

While the elicitation was identical for all groups, the specification was created using different methods and tools, also leading to differences in the students' subsequent validation sessions. The method that was to be used was assigned randomly: three groups used a UML modeling tool,[1] another three groups modeled their findings more flexibly using pens and paper, and the remaining three groups relied on our research prototype. The research prototype that was provided facilitated the discussion between the stakeholder and the students, but required the students to participate as the other roles identified in the scenarios under investigation. Thus, without simulation capabilities, all interactions had to take place directly between users of the system – one stakeholder and

[1] Visual Paradigm for UML, `http://www.visual-paradigm.com/` (accessed June 2013)

the four students in the group. This ensured that we were able to focus the evaluation on the stakeholder interactions without having the simulation interfering with the students.

After each phase, the students answered questions using a five-point Likert scale (1 for *agreement* and 5 for *disagreement*) or free text. Figures 8.2a and 8.2b illustrate how the students rated their results and activities *after the specification phase*, while the results of the questionnaire *after the validation* are presented in Figure 8.2c. For each of these specification and validation methods, we present the responses of twelve students. Thus, the results are sufficiently convincing concerning the engineers' point of view on our approach, while the stakeholder's perspective can only be considered as exploratory (n=3 per group). All results are based on the individual perception of the students and stakeholders.

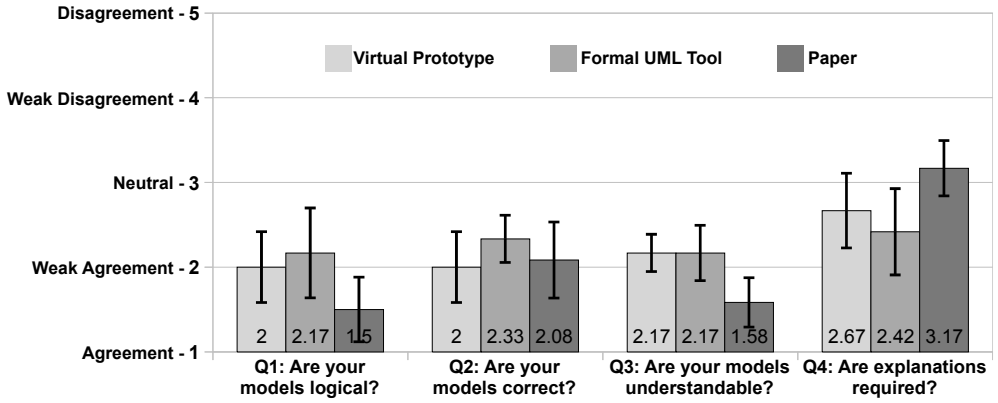
Due to the duration of up to four hours, the dates of the individual sessions were directly arranged between the students and their stakeholder – still, all sessions were conducted within one week during the semester. For these sessions, the students brought their own laptops – depending on the method to be used by the students, we provided them either with pen and paper, or a server hosting our research prototype. All sessions were supervised by two researchers to keep the setting as stable as possible.

Results: When asked *after the specification phase*, all students generally agreed that their results are logical, correct and understandable. Still, we see that students modeling informally (*Paper*) were significantly more confident that end users would be able to understand their results (Q3 in Figure 8.2a with $p < 0.02$ compared to both alternatives[2]). According to the students' responses to Q5, all students were generally rather satisfied with the results they produced during the specification phase. As anybody who has ever used a modeling tool will understand, we were surprised to see that students relying on such a tool had slightly more fun than the other groups.

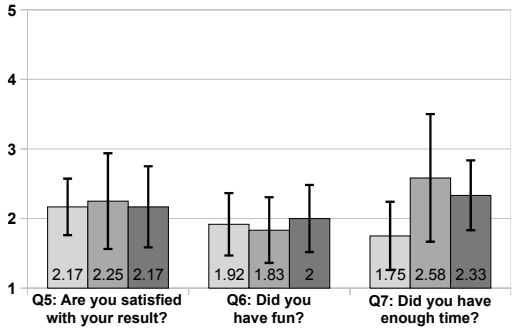
Interestingly, students using virtual prototypes agreed that the time frame of 100 minutes allocated for specifying their findings was sufficient (Q7 in Figure 8.2b). The usage of a formal modeling tool proved to be quite time consuming in comparison. When asked how much more time they felt was necessary, only one of the students using virtual prototypes asked for 20 more minutes. While four students relying on informal representations felt they needed 36 minutes more on average, 50% of the students employing the formal modeling tool required more time ($\bar{x}$=80 minutes). Furthermore, as we also reported in [GGS12], the students using virtual prototypes achieved a much better common understanding of how the stakeholder's company works.

The stakeholder's ability to understand the presented specifications was measured indirectly through the amount of comprehension related questions (Q8 and Q9 in Figure 8.2c). The students presenting their specifications encountered significantly more questions when they relied on the formal modeling tool as opposed to the other two methods

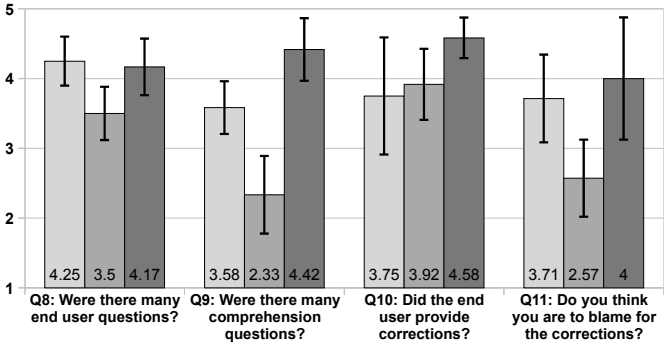
[2] The individual results of the *t-tests* used to evaluate the differences between these three methods can be found in Table B.1.



(a) After the specification, the students rated the quality of their specifications



(b) Questions about the students' modeling activities



(c) After the validation, the students were asked to rate their results

Figure 8.2.: How modelers perceived virtual prototypes compared to a UML modeling tool and using pen and paper instead (n=12; adapted from [GGS12])

($p < 0.05$ compared to both alternatives for Q8). Also, the three different modeling approaches all differed significantly in terms of the amount of comprehension related questions the students had to answer ($p < 0.02$ compared to both alternatives for Q9). While the informal pen and paper approach was perceived as most understandable, the UML models were hardly understandable without explanations. Virtual prototypes, on the other hand, were ranked in between.

When we asked the students whether the end users provided corrections (Q11 in Figure 8.2), the differences were not significant. However, we were surprised to see that the students relying on the formal modeling tool tended to feel responsible for the errors that were made during the specification phase. Our working thesis for this phenomenon is based on the inability to *preserve ambiguity* [Mai12] when specifying findings using a formal model. Multiple times, the students had to decide, for example, whether to model an interaction as synchronous or asynchronous, since there is no *in between* state or *TBD* (To Be Determined) in a formal specification. Consequently, without

sufficient information, deciding for either option put the students at risk of modeling the interaction incorrectly. While this might provoke stakeholder feedback correcting such wrong assumptions, the stakeholders still have to understand the models in order to be able to recognize such errors. The results of Q11 in Figure 8.2c give us a strong indication that the ability to preserve ambiguity in models is quite important. Still, the ability to specify insights without ambiguity leaves the designers more confident in their models. Without any distinction as to how confidently a part of the model was created, revisiting this model later on may lead the modelers to consider all implicit or ad hoc decisions to be just as correct as the rest of the model. This also explains why students using the UML tool were in agreement about their model but still felt rather guilty about the errors that were corrected. While the students relying on the UML modeling tool already had experience in using it, the students using a virtual prototype were initially introduced to it during the evaluation. Hence, we expect better results from settings in which the requirements engineers have used virtual prototypes before.

As for the stakeholders who were exposed to a virtual prototype: all agreed ($\bar{x}$=1, s^2=0; n=3) that what they ended up with after the validation session was co-created by the students as well as the end user. While end users presented with informal specifications felt similarly involved ($\bar{x}$=1.66, s^2=0.58), groups exposing the end users to UML models did not involve them as much ($\bar{x}$=3, s^2=2). Although there is an indication that more flexible approaches can be employed more interactively, i.e. pen and paper as well as our workplace prototype, this small sample of three stakeholders per method is not statistically significant.

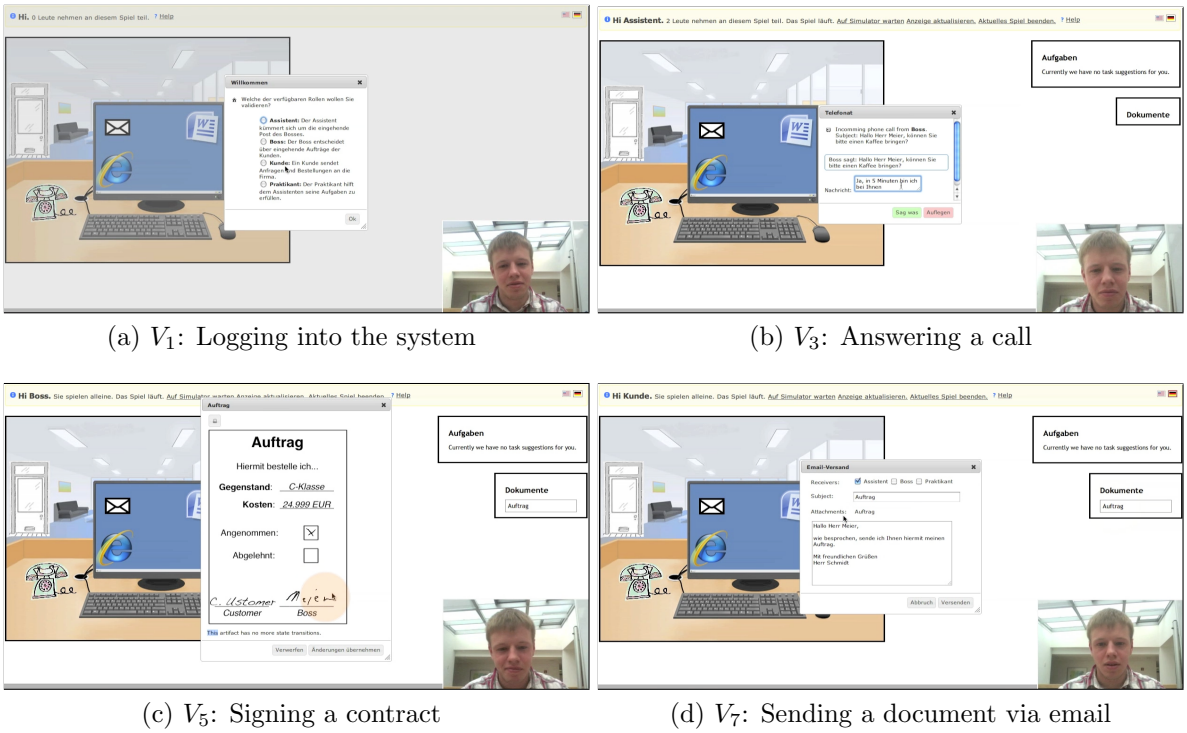
Threats to Validity (based on [WRH+00, MDF05]): Apart from the internal threats to validity discussed in Section 8.1, other threats which apply for this specific setting have to be considered as well. Through the separation of the individual groups during the conduct of the experiment, the threat of unintentionally exposing any of the groups to the treatment of another group, i.e. *imitation of treatment*, has been eliminated. Since all groups were handled equally as ensured by the two supervisors, there was no *special treatment* involved. While the assignment of the students to the different groups was not randomized as part of the experiment, the assignment of the methods and stakeholders for each group was random. Thus, a *biased selection* has been excluded as much as possible for the described setting.

The external validity of our results, on the other hand, requires the results to be *generalizable* across people or situations. By relying on undergraduate software engineering students, we chose prospective requirements engineers to evaluate our approach. Since they are still relatively young and inexperienced, more experienced requirements engineers might perform differently, i.e. better, with either one of the methods. Still, the way the stakeholders became engaged during interactions allows us to conclude that this approach of engaging stakeholders with formal models may also improve if used by experts. What we cannot exclude, however, is the possibility that older stakeholders, or

stakeholders who are not as familiar with computers might react differently when they are exposed to our approach (similar to the effects reported by Sellen et al. [SML+09]). As described above, the setup of our experiment was logistically challenging and quite complex. Thus, a *replication* requires a set of stakeholders who experienced the same scenarios to provide equivalent requirements for all groups. Still, the scenarios related to an online supermarket involve multiple stakeholders and are a representative example of the scenarios for which our approach was created. Consequently, we are confident that replications covering similar collaborative scenarios would lead to equivalent results.

8.2.2. Stakeholder Interpretation of the Virtual Prototype

Interacting with a research prototype can be error-prone and lead to unforeseeable complications. Consequently, for this second experiment the domain-specific visual metaphors employed in the visualization were used in interactions which were captured on video (four snapshots are illustrated in Figure 8.3). Then, these videos were replayed for 15 students without a software engineering background who were asked to interpret the interactions they saw.



(a) V_1: Logging into the system

(b) V_3: Answering a call

(c) V_5: Signing a contract

(d) V_7: Sending a document via email

Figure 8.3.: Screenshots of four of the videos of a student interacting with the prototype

Setup: For our research prototype, we chose specific visual metaphors such as a door to visit someone or a telephone offering an instant messaging window to represent calls in

order to enable stakeholders to experience virtual prototypes. To evaluate whether these metaphors can be intuitively understood, we conducted a study with 15 students (all of them without a computer science background). We prepared seven videos (duration between 13 and 54 seconds) in which a student interacted with the user interface to fulfill seven distinct tasks as listed in the 2^{nd} column of Table 8.1, e.g., reading an email after getting notified about it or accepting a phone call and chatting about a coffee break.

For each video, the 15 students were asked to provide a free text interpretation of what they thought happened in the video. Additionally, they rated their confidence in their interpretation using a seven-point Likert scale (1 for *very uncertain* and 7 being *very certain*). To ensure that the free text responses can be compared, an unrelated introductory video was shown and example answers illustrated the level of detail expected in the free text answers. Then, each of the seven prototype videos was shown twice to the students. After a video was shown twice, the students had as much time as they needed – the next video was played twice once they had all stopped writing.

Furthermore, four undergraduate students from our institute evaluated and compared the free text interpretations to ensure an objective rating of whether each individual interpretation was suitable. For instance, they rated whether "Boss accepts and signs Contract" (*"Boss bewilligt und unterschreibt Auftrag"*) and "Document is signed" (*"Dokument wird unterschrieben"*) adequately summarize the content of the fifth video. Their rating was quite strict as can be seen in Table B.2 along with a German summary created by these four students for each interpretation.

#	Description of the Video	Recognition	Confidence (1–7)
V_1	Logging into the system	93.3% (14)	$\bar{x}$=5.36, s^2=1.48
V_2	Receiving and reading an email	80% (12)	$\bar{x}$=5.67, s^2=0.42
V_3	Receiving and answering a call	100% (15)	$\bar{x}$=6.00, s^2=0.77
V_4	Creating a document	100% (15)	$\bar{x}$=5.27, s^2=1.21
V_5	Interacting with a document	100% (15)	$\bar{x}$=5.73, s^2=1.35
V_6	Visiting another participant	80% (12)	$\bar{x}$=6.00, s^2=0.91
V_7	Sending a document via email	100% (15)	$\bar{x}$=5.91, s^2=1.45

Table 8.1.: Video evaluation of whether potential stakeholders confidently understand the visual metaphors embedded in our virtual prototype (n=15; Confidence ranges from 1 for *very uncertain* to 7 for *very certain*; adapted from [GGB12])

Results: Out of the 15 students watching the seven videos, 98 out of the 105 interpretations (93.3%) were rated as correct. Hence, on average, each video was correctly summarized by 14 out of the 15 participants (s^2=2). The 4^{th} column of Table 8.1 presents the *confidence* of the students whose interpretation was rated as correct.[3] The worst

[3] The seven incorrect interpretations were as follows with $s_i^m = n$ for subject i's confidence value n for video m: $s_{13}^{V_1} = 7$, $s_{09}^{V_2}$ = (no value), $s_{11}^{V_2} = 5$, $s_{13}^{V_2} = 7$, $s_{09}^{V_6} = 6$, $s_{11}^{V_6} = 5$, and $s_{15}^{V_6} = 6$ (cf. Table B.2)

confidence was related to the video V_4 which illustrated the *word processor* metaphor used to create artifacts – still, all 15 interpretations were considered correct. In general, the overall confidence of the correct interpretations was quite high ($\bar{x}$=5.7, s^2=1.11 for 94 valid values), although the seven interpretations which were considered incorrect were, on average, slightly more confident ($\bar{x}$=6, s^2=0.8 for six valid values).

Threats to Validity (based on [WRH+00, MDF05]): By relying on four students to rate the correctness of the subjects' interpretations, we ensured an objective rating process. Due to the high number of videos for this experiment, we cannot exclude a *testing effect* which might lead to improved interpretations for following videos. Still, as the first video represents the first interaction a stakeholder has with our research prototype, it would be unrealistic to switch their order. Additionally, none of the interactions illustrated in the videos would occur by itself, but rather within the context of other interactions. Furthermore, the interactions and their representations shown in the videos are disjoint in such a way that having seen either one does not necessarily aid the subjects in understanding the remaining videos. Consequently, while we cannot guarantee that no testing effect occurs, we are confident that the subjects' ability to intuitively, i.e. without any prior explanation, interpret interactions presented by our research prototype was not impaired. Since there was only one group, there was neither *special treatment* nor the threat of *diffusion* involved.

By relying on students from disciplines different from software engineering or computer science, we obtained a more representative sample of prospective stakeholders whose scenario might have to be captured using our approach. Again, they were relatively young which might have led to better interpretations than an older sample might have produced (cf. [SML+09]). Still, since our sample produced more than 93% correct interpretations without being briefed about what they were about to see, we are confident that other stakeholders might correctly interpret the representation just as well after getting a brief introduction to the tool. Alternatively, in a more realistic setting, stakeholders may even influence the visualization which is, in the end, built to accommodate their understanding of their domain. In general, we are confident that a *replication* using the same or similar videos of the metaphors used in the visualization would yield similar results for an equivalent sample of young students from different disciplines.

Furthermore, we evaluated one specific visualization covering one distinct domain. Since the understandability highly depends on the employed metaphors, results may be different for other domains in which the metaphors have to be more complex. Still, as long as these metaphors are intuitive, stakeholders will understand them.

8.3. Prototype Iterations: Quick and Inexpensive

To be able to *quickly* and *inexpensively* iterate the collaborative scenarios which can be replayed and complemented within the virtual prototype, it should be automated in

such a way that as many tasks as possible are performed unattended, i.e. without a requirements engineering having to intervene most of the time. Still, even before the simulation can be started, it has to be feasible to conduct multiple sessions without delays due to performance issues, especially while loading the models. The costs of setting up the research prototype are illustrated in Section 8.3.1. Then, the automation achieved in our research prototype is described in Section 8.3.2.

8.3.1. Setting up a Simulation Session

To set up a remote interactive session for stakeholders to participate in using the research prototype, its application server (JBoss, cf. Chapter 7) has to be started for the research prototype to be deployed. Then, one of the existing projects has to be picked and its files, i.e. its domain model $\mathcal{D}$, its story patterns $\mathcal{SP}$ and its states $\mathcal{S}$ are loaded. This subsection illustrates the corresponding loading times, i.e. the costs associated with the *creation* of a virtual prototype. All values are averages of 10 repetitions on a *virtual machine* which uses Windows XP SP2 on an Intel Core2Duo@3.0GHz and 1.5 GB RAM.[4]

Starting the Research Prototype on a JBoss: Starting the research prototype on a JBoss (version 7.1) takes $\bar{x}$=6.37 seconds (s^2=1.05). The *deployment* of the necessary files, on the other hand, takes less than a second. Hence, the research prototype can be reached remotely after less than 10 seconds.

Loading the Models Belonging to a Project: After a requirements engineer has chosen the corresponding project or a stakeholder has clicked on a generated invitation link he has received from the requirements engineer (cf. Section 7.2), the data of this project is loaded – this includes the corresponding domain model $\mathcal{D}$ as well as already existing story patterns $\mathcal{SP}$ and all states in $\mathcal{S}$. The concepts collected in $\mathcal{D}$ are limited by the *essential complexity* of the stakeholders' domain [NS00]. Depending on the diversity of the scenarios covered, there may be different final states $s_{term} \in S_{term}$. While the set of states $\mathcal{S} = s_{init} \cup S_{term}$ might increase by one for each new scenario that is covered, the set of story patterns $\mathcal{SP}$ grows more rapidly with each new scenario. For these measurements, elements of $\mathcal{S}$ can be considered as simple story patterns each of which does not affect changes – for a state s_{term}, the corresponding story pattern would be $SP_{term} = (s_{term}, s_{term})$. The initial state s_{init}, on the other hand, would be considered a story pattern $SP_{init} = (\emptyset, s_{init})$. Consequently, for the following measurements, the elements of $\mathcal{S}$ are considered elements of $\mathcal{SP}$.

To load a simple project equivalent to the *Ticket Sale* case study discussed in Section 4.1.2, a domain model extension of eight concepts (extending the 21 concepts within $\mathcal{D}_{basic}$), seven story patterns and one initial state have to be loaded to initialize a virtual

[4] Due to the virtualization, the same measurements can be expected to yield even better results on an ordinary desktop computer.

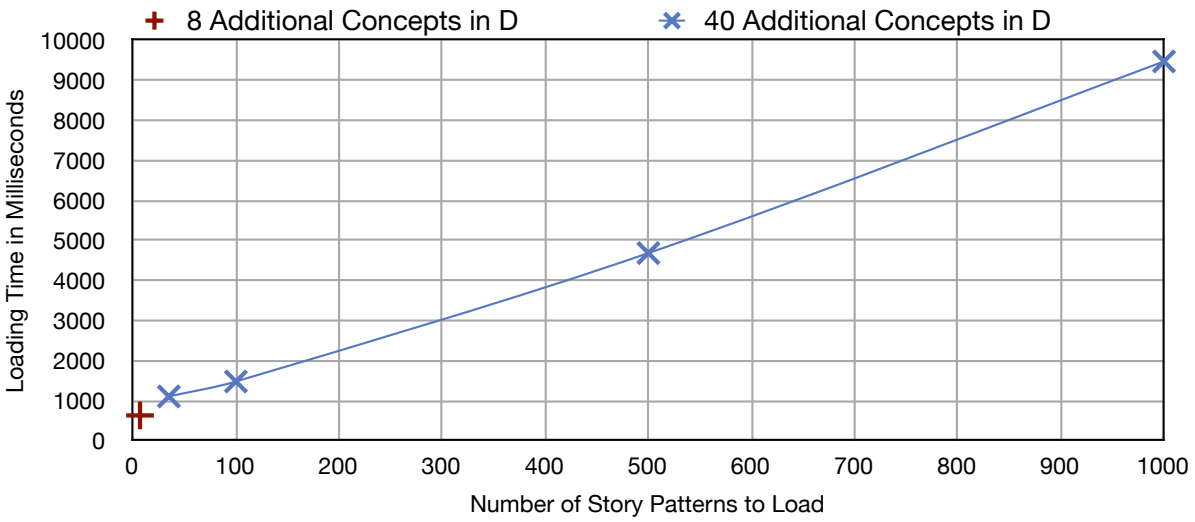


Figure 8.4.: Loading up to 1000 story patterns can be achieved in less than ten seconds (averages of ten repetitions)

prototype. This took, on average, $\bar{x}$=0.631 seconds as illustrated in Figure 8.4. As argued above, a more sophisticated project accumulates more domain concepts, states, and story patterns. Still, even if 40 additional concepts, i.e. 61 concepts in total, are included in $\mathcal{D}$ and up to 1000 story patterns are gathered along the way, initializing a virtual prototype can be done in approximately ten seconds. Thus, starting the JBoss and initializing a virtual prototype using the research prototype can be done quite quickly in less than one minute even for big projects. Additionally, as soon as the research prototype is loaded and initialized, further simulation sessions can be conducted without having to restart the research prototype.

Interactions at Runtime: The *Story Diagram Interpreter* (SDI, cf. Section 7.2) is responsible for checking which of the n story patterns that were already collected can be applied. Measurements related to the SDI and related improvements are out of the scope of this thesis.[5]

When a story pattern is generated, the corresponding pair of states s_{cur} and s_{next} is used to derive the story pattern *in parallel* to the stakeholder interactions. Thus, this generation is done separately and does not impair the overall performance of the virtual prototype.

[5] Concepts such as incremental matching algorithms for increasing the overall performance of the story diagram interpreter as part of our research prototype are discussed and evaluated in the master's thesis of Stefan Kleff [Kle11]. Measurements covering the generation of a look ahead as required for strategies can be found in the [TGRK13].

8.3.2. Automation within the Simulation Loop

The simulation may only be feasibly run, if the requirements engineer's tasks are automated as well during the execution. Figure 8.5 illustrates the automation within the simulation loop. While activities and decisions in blue are executed or determined by the simulator, manual input is required to execute activities in white or determine how to continue at decision points in white.

As long as story patterns which do not belong to a participant are applicable (step c), the simulation may progress (steps d and e) with updates of the participants' visualizations (step f) only. Alongside providing feedback about the specific scenario which unfolds during the simulation, the simulation can progress automatically until one of the following conditions applies:

- a final state s_{term} matches and the participants may acknowledge the completion of the scenario which concludes the session (decision after step g)
- one or more story patterns belonging to a participant are applicable, which enables this participant to choose how to continue between different options provided by the simulator (decision after step j)
- no story pattern is applicable in the current state s_{cur}, which means that a possible continuation is not yet known – a participant may decide to either define a continuation of what she expects to happen (steps p and q) or play in a plausible continuation herself otherwise (steps k, m, and n)

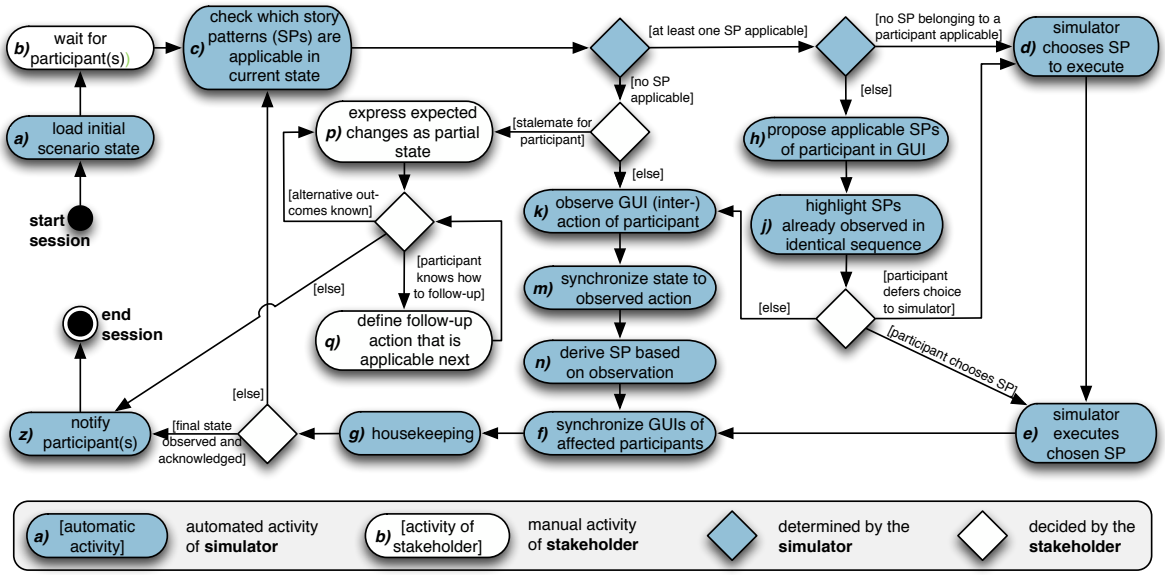


Figure 8.5.: While activities and decision nodes in blue are automated, others require input from participating stakeholders

These are the cases in which the simulator cannot automatically decide what has to happen next due to potentially incomplete information. Consequently, in these cases, the participants have to become active by either deciding what to explore next or complementing the scenario covered in their session. Since everything else is automated, the prototype cannot only be deployed and initialized quite quickly, but also be used semi-automatically to allow stakeholders to complement or validate scenarios – in most cases unattended by the requirements engineers.

8.4. Modifications through Stakeholders

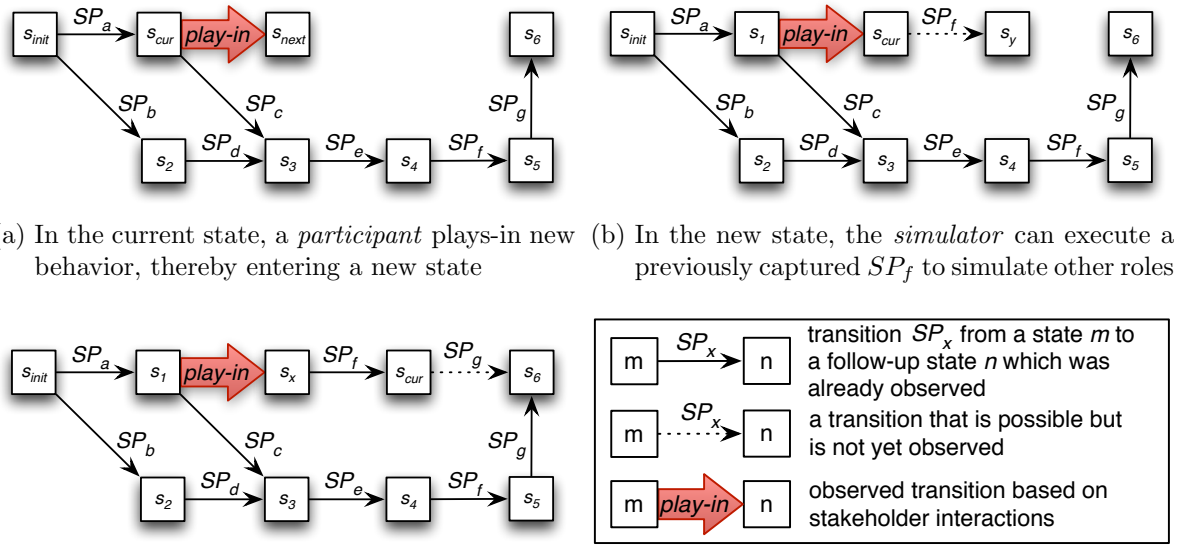
Virtual prototypes of a project differ based on which story patterns have already been collected during the course of the scenario elicitation. By adjusting these models, new scenarios can be explored while other scenarios are excluded. Consequently, the virtual prototype can be adjusted by the stakeholders simply through playing in new story patterns. Stakeholders have two ways of adapting and modifying the set $\mathcal{SP}$ of story patterns which a virtual prototype is based on. First, story patterns can be played in as discussed in Section 8.4.1. Second, each stakeholder may explicitly correct errors in story patterns belonging to her if she recognizes errors, as discussed in Section 8.4.2 (based on [GEHG13]).

8.4.1. Immediate Feedback Based on Play-In

A virtual prototype is derived based on the models ($\mathcal{D}$, $\mathcal{S}$, $\mathcal{SP}$) belonging to a project. Interacting with such a virtual prototype, however, influences its underlying models. Specifically, the set $\mathcal{SP}$ can be changed by adding, removing, or modifying its story patterns. This, in turn, alters which scenarios a participant can experience by interacting with the virtual prototype.

Each time a stakeholder participating as role r plays in priorly unknown and thus unexpected behavior, the simulator enters a new simulation state which might not have been reachable before (s_{next} in Figure 8.6a). This, in turn, may enable other story patterns in $\mathcal{SP}$ which would not have been applicable before (SP_f in Figure 8.6b). Thus, by adding behavior to the virtual prototype and, thereby, re-shaping it, the participant may fulfill the preconditions of story patterns belonging to other stakeholders. Consequently, the execution of such a story pattern changes the state of the simulation, leading to a new current state (s_{cur} in Figure 8.6c). While the former state s_x is different from the new state s_{cur}, it has to be different from the point of view of the participant. Hence, if the simulator executes any sequence of story patterns which lead to a state s_{cur} for which $s_{cur}|_r \ncong s_x|_r$ is fulfilled, the participant receives *immediate feedback* via his GUI on the behavior he just played in – feedback which he might not have gotten otherwise.

Additionally, through the play-in as sketched above, each participant may complement scenarios in which he has to interact with other stakeholders. While *expected continua-*



(a) In the current state, a *participant* plays-in new behavior, thereby entering a new state

(b) In the new state, the *simulator* can execute a previously captured SP_f to simulate other roles

(c) After SP_f was executed, the participant perceives how he was affected if $s_x|_r \not\cong s_{cur}|_r$

(d) Keys for *(a)*, *(b)*, and *(c)*

Figure 8.6.: Participants may receive immediate feedback on new input they provide if existing story patterns become applicable

tions support any scenario complementation as illustrated as part of the case study in Chapter 6, the mechanism sketched above may already suffice for some cases.

In summary, stakeholders can change a virtual prototype by interacting with it. This enables them to receive immediate feedback from other roles through story patterns which are replayed by the simulator. Thus, even though stakeholders can participate individually, the simulator can provide such feedback automatically by design. Relying on these capabilities, participants can complement each others' scenarios step by step.

8.4.2. Stakeholders Correcting Erroneous Story Patterns

As part of his master's thesis, Eichler evaluated the *NL4SP* approach and its NL interface for modifying story patterns [Eic12]. We re-visited the raw data of this evaluation to derive additional statistical interpretations which were also published in [GEHG13].

As soon as a stakeholder perceives an error, i.e. identifies an erroneous story pattern, the stakeholders can directly fix it within the NL4SP prototype. To investigate whether stakeholders understand what story patterns represent in a NL representation, the following hypotheses were evaluated:

H_A Students understand NL representation better than the formal story pattern representation

H_B Consequently, students can also more reliably identify errors in the natural language representation

H_C Additionally, students make more correct modifications editing the NL representation as opposed to the story pattern representation

Setup: To evaluate these hypotheses, we randomly assigned twelve students of disciplines other than software engineering to either group G_1 or G_2. They dealt with the scenario of ordering and receiving an item online which consisted of three steps. These were steps of a scenario they had all experienced before and, therefore, can be considered to be domain experts in. While G_1 answered all questions relying on a formal *story pattern* representation (SP), G_2 used the *natural language* representation (NL). To limit any negative effects an immature research prototype might have, printouts were used for both representations. Thus, the story patterns were presented as images the students had to draw into.

After an introduction (one page) into the representation they had to use, the students started by answering questions about the content of the three steps (H_A). Specifically, they had to identify: what is the value of the order (one property), what does the package contain (two objects) and which conditions have to be met for the order to be shipped (four associations).

Then, the students were asked to identify incorrect elements within the specification of the three steps related to ordering T-Shirts in the representation assigned to them (H_B). For the identification of the five errors that were introduced (e.g., a different *value* for the order and *sunny weather* as additional precondition), they were not allowed to consult the earlier representations.

Finally, the students were asked to specify the postcondition of buying gasoline from a gas station in their respective representation (H_C). For a provided precondition, they had to create or modify five different associations to reach the postcondition. This was done either by drawing and annotating lines in a story pattern printout or by arranging 13 concepts as they would have been proposed by the natural language interface prototype to construct the missing statements, e.g., *you(Customer)*, *from*, and *contains*. In between these tasks, the students filled out short questionnaires concerning the perceived ease of use of their respective method. The students were allowed to take as much time as they needed for these tasks.

Results: It took the students working with the story patterns $\bar{x}$=20.81 minutes (s^2=13), while the other students needed $\bar{x}$=29.1 minutes (s^2=64.7). While the difference is significant ($p = 0.044$), it is consistent with expectations [LS87]. Table 8.2 presents the results per representation and hypothesis. As hypothesized, students working with the NL representation answered significantly (H_A, $p < 0.01$) more understandability questions correctly (81%) than students using the formal representation (45%, 1^{st} row in Table 8.2).

#		G_1 with **SP**	G_2 with **NL**
1	Average of correct answers to seven understandability questions (H_A, $p < 0.01$)	$\bar{x}$=3.17, s^2=1.47 (45% *covered*)	$\bar{x}$=5.67, s^2=1.21 (81% *covered*)
2	Average of *correct* errors (out of *five*) identified per student (H_B, p=0.034)	$\bar{x}$=1.67, s^2=1.97 (33% *covered*)	$\bar{x}$=3.75, s^2=0.70 (75% *covered*)
3	Average of *false positive* errors identified per student (H_B, p=0.027)	$\bar{x}$=2.83, s^2=2.14	$\bar{x}$=0.5, s^2=0.55
4	Average amount of errors pointed out per student = *correct* + *false positive* discoveries	$\bar{x}$=4.5, s^2=1.87 (37% *correct*)	$\bar{x}$=4.25, s^2=0.53 (88% *correct*)
5	Correct modifications out of *five* the students were asked to perform (H_C, p=0.66)	$\bar{x}$=2.5, s^2=1.87 (50% coverage)	$\bar{x}$=3.0, s^2=1.90 (60% coverage)

Table 8.2.: Results are averages of six students working with process steps represented using *story patterns* (**SP**) or *natural language* (**NL**, adapted from [GEHG13])

On average, students using the formal representation pointed out more errors than students working with the NL representation (4^{th} row in Table 8.2). However, out of their answers, errors marked by students using the NL representation covered significantly ($p < 0.04$) more of the real errors (2^{nd} row) and contained significantly ($p < 0.03$) fewer *false positives* (3^{rd} row). In total, while only 37% of the errors marked by students working with the formal representation were correctly identified errors, 88% of the errors marked by the NL group were *real*.

Based on a given precondition, the students had to form statements until a desired postcondition was constructed. Students arranging the 13 partial statements that were provided were more successful in constructing this postcondition (60% coverage of what they needed to describe) than the other group who sketched modifiers and new associations into a story pattern printout (50%, cf. 5^{th} row in Table 8.2). Although students performed better using the NL representation, the difference was not significant (H_C).

In summary, we observed that students without a technical background understood models significantly better in the NL representation (H_A) and were also significantly better at identifying errors (H_B). However, modifying the model and being able to correct such errors was only insignificantly better (H_C).

Threats to Validity: Although the experiment was set up as a *within-subject* experiment [Gre76], the execution of the second treatment was inconsistent in such a way that we were only able to confidently rely on the data of the first treatment. Thus, instead of a n=12 *within-subject* design, we can only offer data for the remaining n=6 *between-subject* results. Consequently, the usable sample is much smaller than expected. Also, due to the small sample size and the students' different academic backgrounds, we cannot exclude effects such as, e.g., chemistry students might perform better with formal models than with NL. Additionally, the experiment we conducted only compared

story patterns to a natural language representation generated from them. Other formal representations might be more or less intuitive for stakeholders. Still, despite the small sample, the statistical tests confirm that the results are significant for H_A and H_B.

The students were assigned randomly to either group which eliminates the threat of a *biased selection*. The design of the experiment also eliminated the threat of *imitation of treatment*, i.e. unintentionally exposing any of the groups to the treatment of another group, since all subjects worked with either representation. Still, while they started out working with different representations, no *special treatment* of either group was involved.

The subjects we recruited for the experiment were students from disciplines other than software engineering or computer science. Thus, while the sample was small, it was still representative for prospective stakeholders using our approach later on. The scenarios covered during this experiment can be assumed to be known to most students – a fact which makes us confident that results from a replication would be quite similar. Still, once more, older people might not be as familiar with such scenarios which might lead to different results.

8.5. Chapter Summary

Apart from the case studies which we discussed for most of the concepts within this thesis, we also empirically evaluated the most critical steps of our approach – namely the ones in which stakeholders directly interact with our research prototype. Using this research prototype, we were able to positively test whether stakeholders would intuitively understand the visualization we provided ($\mathbf{TG}_{2A}$). Moreover, the models underlying the virtual prototypes derived from our research prototype can also be modified indirectly ($\mathbf{TG}_{2B}$). Further, we measured how much time it takes the requirements engineers to set up a virtual prototype for stakeholders to interact with and discussed the degree of automation that is involved in running a simulation session ($\mathbf{TG}_{2C}$). Based on our results, we can confidently state that such derived, model-based prototypes can be employed feasibly to iterate the underlying models. Hence, $\mathbf{TG}_2$ can be considered fulfilled.

Concerning the individual setting in which these stakeholder interactions are to take place: Although the stakeholders can participate decoupled from each other, they are still able to complement each others' scenarios or describe expectations of what needs to happen ($\mathbf{TG}_{1B}$). The state-based simulation of those collaborative scenarios the stakeholders are involved in enables a sophisticated replay of previously observed actions. Consequently, the possibility of replaying these models allows the simulation of possible *reactions* which might be suitable continuations for the participant who is exposed to immediate feedback ($\mathbf{TG}_{1A}$, for cases in which the simulator reaches a state s_x for which $s_x|_r \ncong s_{cur}|_r$ holds). Thus, $\mathbf{TG}_1$ can also be considered fulfilled.

In conclusion, similarly to how physical prototypes are perceived, our evaluation illustrates how our approach brings models closer to being *tangible* for stakeholders and, thereby, provides the advantages of group sessions in individual sessions.

9. Related Work

Our approach of deriving virtual prototypes combines different existing techniques some of which were used in related approaches, in a novel way. In particular, our simulation loop consists of the play-out of formal models, allows stakeholders to play in new models, and further enables stakeholders to define expectations which can the be resolved systematically – all within an intuitive representation. Similar simulation capabilities, which are essential for creating a dynamic and *interactive* animation, are quite common in requirements engineering tools [SRB+00]. However, our approach relies on the synergies between formal models (which inhibit stakeholders from understanding them), domain-specific animations of the changes encoded in these formal models, and a simulator capable of executing these models to simulate identified roles and, thereby, enabling an interactively animated simulation.

It is important to note that stakeholders are an invaluable sources of knowledge, but only as long as they provide feedback within their individual application domain. Thus, presenting them with formal requirements models reduces their understanding of the content, thereby inhibiting their feedback. As motivated in $\mathbf{TG}_{2A}$, our approach focuses foremost on a representation which stakeholders can understand intuitively. Since most other approaches simulating formal requirement models merely visualize the simulation's state within their formal notation (e.g. Seybold et al. [SMG04, SMG06]), we also focus the discussion of related approaches on those which provide an intuitive animation (i.e. **animated play-out**, cf. 2^{nd} column in Table 9.1) within the stakeholders' domain of expertise. Although some of the related approaches offer animated play-out capabilities, only few approaches allow stakeholders to directly **play in** new specifications. The 3^{rd} column of Table 9.1 illustrates which kind of play-in support the related approaches offer.

Moreover, our approach explicitly supports the elicitation and validation of collaborative scenarios, including support for explicitly dealing with expectations and the inherent scenario fragmentation encountered in such scenarios. While some related approaches limit their focus on defining the input and output of reactive systems, others explicitly allow the distinction between different stakeholder groups to model their interactions. The 4^{th} column in Table 9.1 indicates whether support for **collaborative scenarios** is provided.

The **specification techniques** employed by the respective approach are presented in the 5^{th} column of Table 9.1. For all of the related approaches, the feasibility of iterating a formal specification depends on the onetime effort required to adapt the approach to a specific domain and the costs involved in creating the next iteration. Since not enough

information concerning these costs is available, the coverage of $\mathbf{TG}_{2C}$ is only discussed where applicable.

After the related approaches which this chapter discusses are presented in Table 9.1, Section 9.1 focuses on those approaches that provide animated play-out capabilities. Then, Section 9.2 presents related approaches capable of automatically deriving models based on observations of one or more actors. Finally, Section 9.3 discusses the advantages of using transitions between concrete states over message-based scenario specifications, before Section 9.4 summarizes the related work and this chapter.

9.1. Animated Play-Out

While different visualization and representation approaches exist which assist requirements engineers to manage and verify their specifications (cf. Gotel et al.'s overviews in [GMM07, CLGG09]), there are only few approaches which allow *stakeholders* to directly be involved in the validation of *their* (formally specified) requirements. Most of these latter approaches are dedicated to a distinct modeling notation, e.g. for *Z* specifications [ÖPMS98, Tey02], for *Goal-Oriented* specifications [VLMP04, PBM+05], for *Graph Transformation Systems* [EB04, EHKZ05], or for *ScenarioML* specifications [ATB06]. Further, while some of the animations are derived automatically (e.g. [ATB06]), most animations are manually defined and are too specific to be reused in any other projects (e.g. [ÖPMS98, MNK+07]). The remainder of this section discusses such related approaches in detail.

One of the main contributions of **Harel and Marelly**'s Play-approach [HM03a], is the possibility of not only replaying captured system behavior (play-out), but the possibility to capture additional system behavior (play-in) and, thus, new scenarios while doing so. Their approach, however, is centered on user interfaces – for each input the user provides, the play-in covers only how the *system* or its (visible) components should react. While this suffices to capture the interaction between individual stakeholders and a software system, the elicitation and validation of interactions between different stakeholders are out of scope of the Play-Engine. Later on, the approach was extended to cover military settings and present these in 3D environments. This extended the Play-Engine to cover a limited set of scenarios in which actors move and shoot [AH07], or react on external stimuli by performing specific actions [HSKS08]. In both cases, however, the focus is on training exercises during which all actors are essentially doing the same (trying not to get shot or following a team leader, respectively) – no *collaborative* behavior is explored. Furthermore, the perception of individual actors is purely event-based – hence, a manual mapping between GUI and events needs to be established as well as whether an actor is visible to another one [AH07].

Approach	Capabilities of the Approach: Animated Play-Out	Play-In of New Specifications	Collaborative Scenarios	Specification Technique	Description
[HM03a] [AH07] [HSKS08]	●	● (explicitly defined)	◐	*Live Sequence Charts*	play-in and smart play-out based on GUI mock-up, later 3D animation
[VLMP04] [PBM+05]	●	○	○	*Finite State Machines*	focus on validation of goals related to control systems
[KNNZ00]	●	○	◐	**GTS**	interactively animates graph-based specifications
[EB04] [EHKZ05]	●	○	◐	**GTS**	editor allows to create animations for graph transformation systems
[MPGK00]	●	○	◐	*Labeled Transition System*	animation steps are assigned to transitions in timed automata
[UCKM04]	●	○	●	*Labeled Transition System*	interactive animation of play-out, manually defined partial states
[ATB06]	●	○	●	ScenarioML	automatically derives animation for manually specified scenarios
[Sch07]	●	◐ (informal)	◐	Use Cases	GUI mock-up-based replay of use case steps with informal play-in
[GH06] [VBI+09]	◐	◐ (pattern recognition)	●	*Messages between users*	focus on negotiations, behavior specified manually
[DEVG08]	●	● (based on observations)	◐	Segments of "action sequences"	smart play-out of *movement* for virtual characters played in via joystick and mouse
[GMM09]	○	● (based on observations)	○	**GTS**	automatically infers specification of *Java classes*
[Aal07] [Aal11]	○	● (based on observations)	◐	Petri nets	automatically derives process models from *event logs*
Virtual Prototypes	●	● (based on observations)	●	**GTS**	

Table 9.1.: Overview of related approaches focused on which features they provide ("●" for full support, "◐" for limited support, and "○" for no support) and their specification technique ("**GTS**" for approaches relying on graph transformations, and *italics* for message-based approaches)

Van et al.'s approach [VLMP04] provides animated play-out capabilities for the validation of *goal-oriented requirements* (cf. [Lam01]). Hence, it is based on goals and their operationalization, i.e. their preconditions and prescriptive postconditions. By using operationalized goals, *goal-based state machines* are generated which can simulate (parts of) the behavior required to achieve these goals. Moreover, if these goals contain temporal logic specifications, so-called *claim state machines* can be generated. These, in turn, are used to monitor their goals and their respective goal state machines during the simulation and animation. To animate the simulation, a mapping is created which assigns a picture to each state in which such a goal state machine can be. For the simulation, stakeholders can provide input through "input event panes" [VLMP04] or "domain-specific control panels" such as levers found in a train [PBM+05]. Based on the mapping between animation and state machines, these inputs modify the state of the underlying state machines which, in turn, leads to an update in the visualization. Thus, this approach focuses on the validation of reactive behavior based on goals which have to be achieved or maintained. Although multiple different views are supported, the effort necessary to create an instance situation to iteratively identify errors and fix them is considered to be high due to a rather restrictive visualization engine. Finally, this approaches is focused on the validation of control systems – collaborative scenarios cannot be elicited or validated.

The models employed by **Köhler et al.**'s approach [KNNZ00] are quite similar to the ones we use. While we employ story patterns, they employ *story diagrams*, which consist of story patterns and Java statements, with control flow in between them [FNTZ00]. Based on the *FUJABA* environment [FNTZ00, NNZ00], a specific state the system can be in during a simulation can be visualized using a *Dynamic Object Browsing System* (DOBS), which represents this state along with iconic representations of these objects. Their approach is illustrated using a production control system, which is specified, simulated, and animated. Stakeholders may interact with this animation via DOBS which offers all applicable methods defined for each of the classes of the system.

Similar to our approach, the behavioral models *belong* to specific classes – in our approach, only roles are valid targets. Consequently, in both approaches, behavior can be assigned and, hence, allows to model collaborative scenarios. However, although the play-out is animated, the interactions between stakeholders and the animated replay of the story diagrams are reduced to invoking methods based on their signatures. After selecting an object, its methods are displayed and can be invoked by the user, e.g., after selecting a `shuttle`, among the options are `checkPlan()`, `getAt()`, `doProduce()`, and `toString()`. Unfortunately, such an interaction is not as intuitive for stakeholders, since these signature names are based on conventions stakeholders are typically not aware of. Furthermore, the approach does not enable stakeholders to play in new behavior.

Ermel et al.'s approach extends a graph transformation system with additional elements for which graphical icons and animations can be assigned [EB04]. Their approach

explicitly supports the definition of different domain-specific scenario views. Moreover, all graph transformations are extended to include these graphically representable elements as well. Since their rules are quite abstract, they can be visualized for different scenarios by annotating the models with different visualizations. They used this approach to create animations for graph transformation systems derived from UML specification [EHKZ05]. According to Al-Rawas and Easterbrook [ARE96], their visualization approach might be considered as a *self-explanatory annotation* of a formal model. In essence, while their approach tries to provide specific visualizations and animations for generic rules, our approach provides a generic visualization that is suitable for a complete domain for specific rules. While their animation may support the validation of collaborative scenarios by animating them, perspectives of different roles seems only viable by creating and defining distinct scenario views which only represent parts of the current state of the simulation. Further, their approach works only in one direction: it can only visualize state modifications. Without *affording* stakeholders to modify the state from within their animated view, Ermel et al.'s approach does not provide play-in capabilities.

Magee et al. pointed out that animations which are clearly separated from the specification that is to be animated can be considered *annotations* of this specification [MPGK00], similar to how Haumer et al. annotated specific goals of a goal model with videos illustrating their meaning [HPW98]. Magee et al.'s approach [MPGK00] is based on multiple *labeled transition systems* (LTS) which are "augmented with a finite set of (real-valued) clocks" to derive *timed automata*. These automata represent interacting components and their respective states. Each transition within such a timed automaton may be annotated with a graphical animation which may illustrates this transition in the "problem domain", i.e. in a way that is understandable for stakeholders. These automata (or subsets thereof) may be initialized and started to investigate their behavior, i.e. the specification can be played out. Further, the modelers can provide stakeholders the possibility of interacting with the animation by setting specific *conditions* using hard-coded buttons in the GUI. Although the approach allows the replay of specific sequences of behavior, it is limited to the play-out and, hence, validation of specifications without any possibility to amend or modify the underlying automata.

Uchitel et al. propose an approach which provides animated play-out for collaborative scenarios [UCKM04]. However, they point out that scenarios carry only implicit states, especially if these scenarios are specified by *message sequence charts* (MSC) that represent the interactions between components based on the messages these components exchange. Based on scenarios specified via MSCs of how components and roles (i.e. users of the system) interact, an LTS can be derived for each of the involved actors.

By simulating these specifications, a stakeholder can participate as one of the roles interacting with the specified system. However, as pointed out, their approach does not define an overall state for such a simulation. Instead, each LTS of each component possesses a distinct internal state by itself. In order to interactively visualize the simu-

lation for the stakeholder, Uchitel et al. define "abstractions of system states" referred to as *fluents* which are based on "the occurrence of events such as those that appear in operational scenarios".

The visualization and animation is presented using an interactive web page. This web page contains elements (buttons and hyperlinks) which enable the participant to invoke transitions which are possible in the corresponding LTSs. These transitions conform to the message exchanges defined for the participant's role in the scenario specifications.

Fluents are assigned to each GUI element that can change during the simulation. Thus, based on which events have already occurred and, hence, the state of the fluent, specific parts of the animated visualization are either active or not. Since these elements also indicate which actions the participant may perform in accordance with the simulated LTSs, a participant may point out errors based on the availability of such actions (or their corresponding GUI elements).

To summarize their approach, Uchitel et al. provide a web-based representation which is interactively simulated based on LTSs derived from scenario specifications. Their usage of fluents and conditional elements in the web-pages that are generated enables them to provide distinct perspectives on the current state of the simulation – similar to a participant's view $s_{cur}|_r$ in our approach. However, they introduced an additional layer of abstraction between the simulation and its visualization. Consequently, as they point out themselves, each time a participant points out an error, it may always be caused by an incorrectly defined fluent. Thus, they first have to check whether the manually created "visualization criteria", i.e. the abstract states defined by the fluents, are correct. In our approach, on the other hand, such a mismatch can be investigated directly based on the current state s_{cur} of the simulation – especially to find out why a specific story pattern was not applicable yet (cf. Section 4.2.1). Moreover, through its emphasis on validation, this approach does not allow participants to play in additional scenarios.

In **Alspaugh et al.**'s approach [ATB06], scenarios are created (manually) as structured text. Then, an animation is derived automatically using animated characters (representing people, actors, or other entities) which move toward each other when they interact. However, each scenario is only replayed "whole" without the possibility to explore overlapping or alternative situations. While the approach allows the specification and animation of collaborative scenarios, the result is equivalent to a video without any interactive components. Although Brill et al. [BSK10] emphasize the value of "videos as a means of documenting requirements" as they are "more concrete and easier to understand by stakeholders", stakeholders cannot amend, complete, or directly correct any of the scenarios which are replayed.

Schneider's *Fast Feedback* technique [Sch07] allows requirements engineers to iteratively elicit and validate requirements in individual sessions of up to three stakeholders by combining use cases defining scenario steps with UI mockups covering these steps. By not only asking for use case information, but also for user interface information, the

requirements engineer can sketch corresponding UI mock-ups which are then automatically connected to steps in these use cases. A tool running on a tablet-PC, which is used to capture all information, continues to "generate an animation of the pencil-prototypes" which stakeholders can interact with by pretending to enter data. The stakeholders' actions, i.e. scribbling values in forms or adding annotations to the mock-up, are recorded and provide "information on how the stakeholders intend to use the system". While the annotations and scribblings of the stakeholder appear on the mock-up, the GUI is not interactive, i.e. it does not react to stakeholder inputs.

Similar to the Play-Engine [HM03a], scenarios are played out and animated using GUI mock-ups. Additionally, stakeholders can directly experience and validate these mock-ups, while their informal input is recorded. With its emphasis on getting as much information as possible from the stakeholder, this approach requires that requirements engineer manually integrates this input consistently with requirements of other stakeholders or stakeholder groups. Therefore, while the *Fast Feedback* technique allows stakeholders to express their thoughts and intended usage patterns centered on UI mock-ups, it provides only limited support for analyzing the recorded information and coping with complex collaborative scenarios.

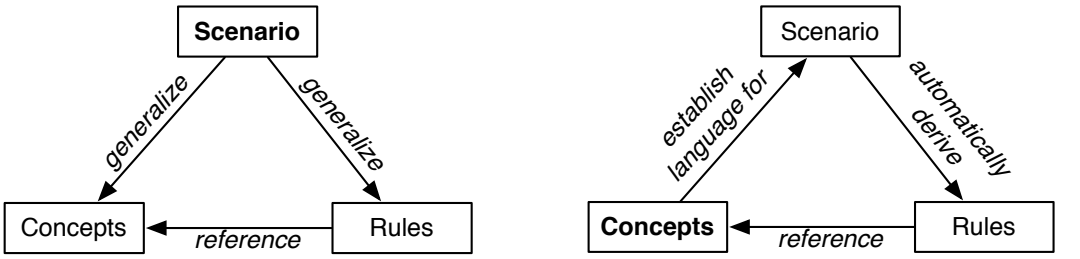
Guyot and Honiden [GH06] provide a manually created animation of collaborative scenarios in which multiple participants compete against each other in trading scenarios, to elicit and understand the strategies or rationale behind the participants' behavior. In their approach they even enforced a distribution of the participating stakeholders over different rooms, to ensure that all communications between them took place within their tool. Only then were they able to record all interactions and query the participants based on events from these records in follow-up debriefing sessions. Further, these records can be used to "automatically extract interaction patterns" from the participants' observed behavior. Then, these patterns must be *manually* implemented in order to mirror participants' strategies or to provide suggestions for participants in subsequent sessions. This approach was extended by Vasconcelos et al. [VBI+09] with the intent of introducing "artificial players" which base their behavior and decisions on observed communication of human players. However, they also acknowledged that they have not yet resolved how to *automatically* generate behavior based on such observations.

Dinerstein et al.'s approach [DEVG08] proposes *programming by demonstration* to allow end users to play in behavior by moving virtual characters or entities such as humans, animals, and submarines through virtual environments. The observed behavior, i.e. *movement* played in through joysticks, mouse, and keyboard, is captured and segmented for later sessions. When the virtual entity has to be simulated, this segmented behavior is "combined in novel sequences to create new motions" to explore all alternatives of how this entity should continue. To find the best alternative, a *planning tree* of a predetermined length `n` is generated and scored based on a *fitness function*. Such a planning tree can be considered equivalent to a look ahead (cf. Section 4.3).

The play-out of Dinerstein et al.'s approach is not only animated suitably for "non-technical users", but it can also be considered smart – based on a predefined or derived fitness function which scores possible alternatives, novel behavior can be explored to find the best alternative. However, their approach is rather restrictive in terms of how a user can interact with their environment apart from moving through it. Any "high-level actions" require different translations and mappings to map a mouse click to a predefined tasks such as taking and placing boxes. Further, their synthesis of observed behavior can only be applied to optimization problems such as the *path planning* examples they illustrate. More complex interactions or collaborations between roles *with different tasks* are not discussed in their approach.

9.2. Play-In of New Specifications

While not providing an animated representation suitable for stakeholders, other related approaches exist which provide similar automation capabilities for deriving behavioral models based on observations. For instance, **Ghezzi et al.** [GMM09] try to "infer formal specifications of black box components by observing their runtime behavior". For instances of specific Java classes, this is achieved by invoking the methods of the instance, querying its internal state using *observers* (i.e. methods of the class with non-void return types) to distinguish different states the object can be in, and comparing pairs of succeeding states. Based on such pairs of states, the changes in between can be gathered and defined using graph transformations. These, in turn, represent the inferred behavior of the Java class under investigation. While their method of automatically deriving behavior is quite similar, their approach is restricted to a single actor, i.e. an instance of a class.



(a) Heckel starts by observing scenarios and derives the rest (references between concepts and rules are implied)

(b) After establishing a domain model with the stakeholders, they play-in scenarios so that rules can be derived

Figure 9.1.: Comparison between (a) Heckel [Hec06] and (b) our approach

While Heckel argues that the domain concepts can be generalized from scenarios [Hec06], this is unrealistic if multiple or heterogeneous stakeholder groups are involved.

Unlike an objective observer who would deterministically derive the same rules and concepts (cf. Figure 9.1a and Ghezzi et al.'s approach [GMM09]), stakeholders are neither objective nor deterministic. The requirements engineers can only elicit the actual scenarios by establishing a common understanding of the domain concepts (cf. Figure 9.1b), iteratively validating the stakeholders' statements, and correcting misunderstandings.

Van der Aalst proposes an objective approach based on process mining (*ProM*, [Aal07, Aal11]), which is able to derive Petri nets (or other types of process models) of how different people can achieve their goals collaboratively based on log files detailing traces of their collaborative scenarios. However, ProM can only be used if a logging system is already in place. Further, since ProM is only able to use any information that is available in these logs, this logging system needs to be verbose to provide enough information. Even then, analog interactions which do not leave traces in an event-based logging system cannot be captured.

Since humans tend to abstract from details, approaches which generate models based on objective observations are usually closer to reality than models which are created by humans [Aal07]. Thus, while both approaches can uncover the *whole* complexity of all scenarios, no representation suitable for stakeholders is available. Our approach, on the other hand, provides such an intuitive representation and is capable of eliciting this complexity as well.

9.3. Scenario Specifications

Most approaches of synthesizing scenarios start with collections of potentially incomplete, implied (cf. [UKM01, UKM02a], or negative (cf. [UKM02b]) instance scenarios. These are usually specified message-based, e.g. using Message Sequence Charts (e.g. Uchitel et al. [UKM01]), Sequence Diagrams (e.g. Whittle and Jayaraman [WJ10]) or Live Sequence Charts (Harel et al. [Har01]). For scenario specifications containing multiple interacting components, it is feasible to generate either a *global* state machine or a set of communicating *object* state machines (i.e. one for each involved object) [LDD06]. Such a synthesized state machine conforms to all scenario specifications – negative and positive. Liang et al. [LDD06] discuss a total of 21 different approaches of generating state machines from message based scenario specifications.

Still, these approaches are not suitable to elicit human interactions, which are limited by what stakeholders can perceive from their individual context. To obtain behavioral models of what stakeholders do, it does not suffice to know which messages arrived at a stakeholder in which sequence, but rather which artifacts and information they can see, access, and, hence, use to make their decisions. For instance, only by moving to the Notifier's location (SP_{move} in Figure 6.8a) can the Boatman fulfill the Notifier's expectation, that *a lifeguard boat will arrive*, eventually (Figure 6.8b). By eliciting and validating them in a state-based manner, it becomes viable to visualize the current

state a stakeholder is in based on what he has access to. In Uchitel et al.'s approach [UCKM04], this information is also required for the visualization. However, they must define additional abstract states (fluents), since their message based specification using MSCs cannot suitably express the presence of information. In other words, follow-up activities become available based on what an individual participant has already achieved – not based on the history of messages he has sent or received. While message-based abstractions are suitable for the definition of reactive software systems, they are not suitable for capturing concrete stakeholder behavior or exploring the rationale behind it. By being able to distinguish between different artifacts and their states, the requirements engineers are able to specify workflows not only in an activity-based manner but also artifact-based as argued by Nigam and Caswell [NC03].

9.4. Overview and Chapter Summary

To validate their requirements, stakeholders have to understand them. While approaches exist which allow stakeholders to *experience* their requirements through simulation and animation, these approaches are mostly restricted to the design of reactive systems [VLMP04, PBM+05]. Based on a specific stakeholder input, these approaches provide and visualize an output which conforms to the stakeholder requirements and which the stakeholder can understand and validate. However, no approach produces such an animated output based on stakeholder interactions in collaborative scenarios. Furthermore, only few approaches provide stakeholders the possibility to intuitively express their activities, i.e. what they do, how they do it, and with whom they do it, formally.

Our approach, on the other hand, rearranges and replays behavioral specifications automatically derived from stakeholder observations to produce animated output as a response to a specific stakeholder input. Thereby, participants are enabled to experience and validate their collaborative scenarios. Additionally, while some of these advantages can also be achieved in group sessions involving all stakeholders, our approach explicitly supports decoupled sessions with individual stakeholders. These capabilities, in combination, amount to a novel approach of *prototyping* collaborative scenarios which brings the advantages of group sessions to individual sessions.

10. Conclusions

Eliciting and validating requirements is quite complex, especially if many different stakeholders and their usually disjoint perspectives are involved. Multiple formal approaches can support the requirements engineer to cope with the inherent complexity of such settings. For instance, by employing formal models to specify and automatically derive behavioral specifications, different model checking techniques (cf. [SRB+00]) and automated visualizations for requirements engineers (cf. [GMM07]) are supported. However, employing those approaches introduces overhead – especially in interactions with the stakeholders. Since these interactions are essential to gather and validate the stakeholders' requirements, the requirements engineers have to make a trade-off between formal and informal modeling approaches. In this thesis, we presented an approach that offers a better trade-off for the requirements engineer.

Our model-based virtual prototyping approach provides an understandable representation which allows stakeholders to experience, judge, and comment on the formal models which they would otherwise not be able to interpret (covering $\mathbf{TG}_{2A}$). Based on the similarities between prototypes and models, we proposed virtual prototypes to overcome the prerequisite of formal models, i.e. the knowledge required to understand and work with them. To represent these models interactively, they are *played out* – each identified stakeholder can be simulated by executing the corresponding story patterns encoding the behavior observed from representatives of that role. This enables the simulator to provide immediate feedback as a reaction to stakeholder input (covering $\mathbf{TG}_{1A}$). Further, our approach affords stakeholders to *play in* what they would normally do in such a way that formal models can be derived from observing the stakeholders interact with the animated user interface of the simulation (covering $\mathbf{TG}_{1B}$ and $\mathbf{TG}_{2B}$). Both capabilities, *play-out* and *play-in*, were illustrated using a real-life case study of selling movie tickets.

To deal with stalemates, i.e. situations during the elicitation in which a participant needs to wait for a reaction of another role and cannot deterministically predict which answer to expect, we introduced a black box abstraction for yet unknown activities of other stakeholders. Thus, by intuitively defining the different alternative responses a stakeholder expects and how he or she would continue in each case, these scenarios can systematically be explored, played in, and validated in decoupled stakeholder sessions. The definition and resolution of such expectations was illustrated using a real-life example of a lifeguard service.

The concepts were prototypically implemented using Eclipse and EMF. Moreover, based on the resulting research prototype, we evaluated the goals defined for this thesis

– namely whether our approach supports stakeholders in the validation and elicitation of the scenarios they are involved in, and whether our approach supports requirements engineers to quickly and inexpensively iterate story patterns (covering $\mathbf{TG}_{2C}$). In the evaluation, the understandability of our animated domain-specific representation yielded positive results. Furthermore, the costs of deploying and iterating story patterns were evaluated. Overall, our approach fulfills the thesis' goals outlined in Section 1.2, decouples stakeholder interactions, and, thus, enhances individual stakeholder sessions. In conclusion, similar to how physical prototypes are perceived, our approach brings behavioral models describing collaborative scenarios closer to being *tangible* for stakeholders.

10.1. Discussion

In most engineering and design disciplines, tangible prototypes can be used to manifest, communicate, and iterate ideas. For software systems supporting collaborative scenarios, however, prototypes are either infeasible or restricted to UI designs. We presented our approach of prototyping collaborative scenarios by simulating formal behavioral models. Based on its automation capabilities and an understandable representation, the approach allows requirements engineers to inexpensively iterate story patterns with stakeholders. Our approach brings the main advantage of informal requirements – the fact that stakeholders usually understand them without explanations – to formal requirement models. Thereby, we provide requirements engineers with a better trade-off when they have to decide whether to document requirements formally or informally.

Generalizability of the Approach: The approach of employing virtual prototypes in requirements engineering was successfully implemented and evaluated for the domain of collaborative scenarios as they occur in office workplaces. These scenarios provide a rich mixture of formal and informal artifacts, and thus *solid* and *fluid* information (cf. [SS12a]). Furthermore, they provide standardized communication channels which can be covered quite easily using the presented metaphors. The metaphors implemented in the research prototype have been created, tested, and iterated over a period of three years. Thus, while we are confident that such metaphors relying on the stakeholder domain may be found and integrated easily, this aspect has not yet been evaluated for other domains. Especially the notion of *visibility* for a role to reflect its perspective $s_{cur}|_{role}$ may be to general for more complex domains in which the visibility may be computed differently for each role.

Concerning the scenarios which may be covered, it has to be noted that while our approach can convincingly represent media breaks as they often occur in collaborative scenarios, it is less effective in settings in which stakeholders rely extensively on different software systems to share their knowledge and interact with each other. Such systems and their functionality would have to be mimicked within our scenario simulations to allow stakeholders to correctly use these systems as they do in their daily

routine. Otherwise, the elicited scenarios would be different from how the stakeholders actually work.

10.2. Future Work

Apart from additional real-life case studies covering other domains in which the presented approach can be evaluated, further opportunities exist.

Conceptual Extensions

Parallelization of Scenario Execution: Currently, the simulator enforces a strict sequentialization of all steps of the involved roles. Instead, the simulation might be parallelized by finding one specific story pattern to execute for each of the simulated roles. Executing them, still sequentially but seemingly in one step, might be perceived as more realistic, as, e.g., multiple asynchronous interactions can arrive seemingly simultaneously. However, it has to be ensured that each story pattern chosen for one role does not change the applicability of any story pattern chosen for another role. Consequently, it has to be ensured that the graph transformations chosen for simultaneous execution "only share items which are preserved" (cf. *Critical Pair Analysis* [HKT02]). Otherwise, trying to simultaneously execute such story patterns may lead to inconsistent states.

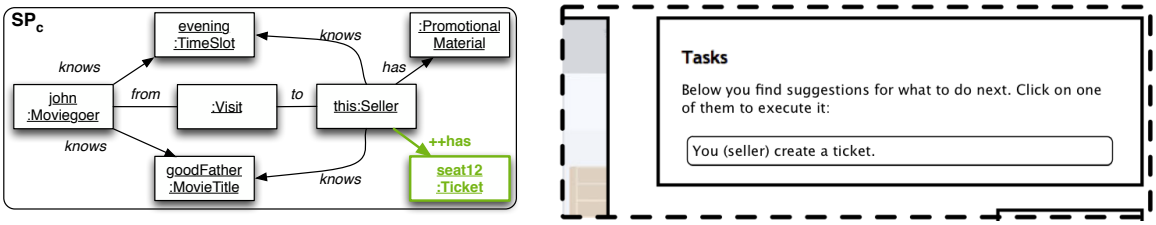
Parallel Scenario Completion: As described in Chapter 6, our approach currently focuses on completing one scenario at a time. The overall number of stakeholder interactions can be further reduced by taking into account how stakeholders interact in multiple scenarios. Specifically, the resolution of triggers defined by different stakeholders might be handled explicitly within the same session.

Deriving User Stories from Collected Story Patterns: As pointed out by Leffingwell [Lef11], user stories are "brief statements of intent that describe something the system needs to do for some user." These usually have the following form:

As a [user role], I can [activity] so that [business value].

Consequently, out of these three parameters, two are already encoded in each story pattern. Since each story pattern belongs to a specific role r, the user role is provided. Furthermore, the *activity* is equivalent to a natural language representation of the post-condition of a story pattern – details of this representation are described in Section 2.5. While the preconditions of the activity are also available, these can be ignored for the user story. The *business value*, on the other hand, is not an explicit element of the story pattern. To obtain this goal, i.e. *why* the activity is executed, two possibilities are imaginable: either the requirements engineer simply asks the stakeholder to fill in this single

remaining gap, or story patterns which were observed subsequently in other scenarios are used to express activities which were only possible afterwards. This is illustrated in the caption of Figure 10.1: only after a ticket is created, may the seller hand it over to a moviegoer (cf. story pattern SP_d in Figure 4.4). Consequently, it may be feasible to automatically extract the missing parameter from observed scenarios and, thereby, generate user stories from the collected story patterns.



(a) Story pattern belonging to the role seller (b) The activity encoded in the story pattern

Figure 10.1.: "As a `seller`, I can `create a ticket` so that `I can hand over ticket to moviegoer`"

Prototyping the Required Software System within the Scenario Simulation: Our approach captures collaborative scenarios by defining and refining the pre- and postconditions of stakeholder activities. Therefore, instead of prototyping design alternatives, we focused on establishing the collaborative scenarios which the required software system has to support. Thus, by introducing the notion of a *software prototype* as an active agent in our approach, requirements engineers may define and iteratively prototype the capabilities of the required software system and how it may fulfill the stakeholders' needs. Quite similar to how stakeholder activities are defined using pre- and postconditions, the capabilities of such a software system may be prototypically defined using story patterns as well. Within our scenario simulation, stakeholders may interact with such a prototype of a planned software system that provides the required capabilities for, e.g. replacing several manual activities a single role has to perform. Consequently, the activities of the prototyped software system could be offered to participants within their intuitive GUI as soon as the corresponding preconditions are fulfilled. Then, participants are enabled to directly comment on whether these prototyped capabilities offered to them fulfill the stakeholders' needs.

Detecting Workarounds in Existing Scenarios: Building on the prior extension, if our approach can be used to iteratively prototype capabilities of software systems, it would also become feasible to use the story patterns from which the resulting software system is built as an *oracle* [Ham94] to test the implementation against.

Furthermore, as of now, process mining relies on event logs [Aal07] to derive process models. However, as Bhattacharya et al. point out [BCK+07], tasks in such process

models "do not model their internal behavior". Instead, their focus is on reasoning "about the composition and results of activities" without "describing what these activities are". Since the captured story patterns model this internal behavior, stakeholders working *around* the realized software system can be recognized by observing the execution of individual activities of this running system. The domain model $\mathcal{D}$ provides the required object level information to compare the *actual* usage of the system against the *expected* usage. Thus, based on such derivations, *workarounds*[1] which may require a modification of the system may automatically be identified.

Research Prototype

Flexible Mapping between Front End and Back End: In this thesis, the approach was implemented and evaluated for one domain. In its current version, the mapping between the options offered in the front end and the corresponding modifications of the current state of the simulation in the back end are reusable, but hard-coded (cf. step *m* in Section 7.2). This mapping can be simplified by utilizing *story patterns* instead: for each option in the front end, the mapping invokes a generic story pattern which leads to the same changes. Each of these generic story patterns would then explicitly represent a *communication primitive* [GH06]. Consequently, the approach is then easier to adapt to other domains, since the requirements engineer would then be able to modify an animation in the front end independently from its corresponding generic story pattern in the back end.

Support for Visual NACs: The version of the *Story Diagram Interpreter* which is used by the research prototype did not support visual NACs in story patterns. To overcome this restriction, these conditions had to be encoded as OCL-constraints, which unfortunately introduced a textual abstraction on top of the graphical one. Since it is planned to support visual NACs in forthcoming versions of the SDI, migrating to an updated version may substitute this workaround.

[1] Poelmans [Poe99] defines a workaround as "opportunistic solution" of stakeholders or "a coping strategy that deviates from the strategies that have been defined in the workflow systems".

Bibliography

[AAF+09] Ateret Anaby-Tavor, David Amid, Amit Fisher, Harold Ossher, Rachel Bellamy, Matthew Callery, Michael Desmond, Sophia Krasikov, Tova Roth, Ian Simmonds, and Jacqueline de Vries. An algorithm for identifying the abstract syntax of graph-based diagrams. In *Proc. of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC'09, pages 193–196, Washington, DC, USA, 2009. IEEE Computer Society.

[Aal07] Wil van der Aalst. Trends in business process analysis: From validation to process mining. In *Proc. of the International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Madeira, Portugal, June 12-16 2007.

[Aal11] Wil van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin Heidelberg, 2011.

[AB07] Ian Alexander and Kent Beck. Point/counterpoint. *IEEE Software*, 24:62–65, 2007.

[ABD09] Ian Alexander and Ljerka Beus-Dukic. *Discovering Requirements: How to Specify Products and Services*. John Wiley & Sons, 2009.

[AG01] Vincenzo Ambriola and Vincenzo Gervasi. On the parallel refinement of NL requirements and UML diagrams. In *Proceedings of the ETAPS 2001 Workshop on Trasformations in UML*, Genova, Italy, 2001.

[AH07] Yoram Atir and David Harel. Using lscs for scenario authoring in tactical simulators. In *SCSC: Proceedings of the 2007 summer computer simulation conference*, pages 437–442, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[AHK13] Abdullah Alshanqiti, Reiko Heckel, and Tamim Khan. Learning Minimal and Maximal Rules from Observations of Graph Transformations. In *Proc. of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques*, GT-VMT'13, 2013.

[AHTG11] Niaz Arijo, Reiko Heckel, Mirco Tribastone, and Stephen Gilmore. Modular performance modelling for mobile applications. In *Proc. of the 2nd joint WOSP/SIPEW international conference on Performance engineering*, ICPE'11, pages 329–334, New York, NY, USA, 2011. ACM.

[Ale05] Ian Alexander. A Taxonomy of Stakeholders: Human Roles in System Development. *International Journal of Technology and Human Interaction*, 1(1):23 – 59, 2005.

[Ale11] Ian Alexander. GORE, SORE, or What? *IEEE Software*, 28:8–10, 2011.

[AM04] Ian Alexander and Neil Maiden, editors. *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley, New York, August 2004.

[ARE96] A. Al-Rawas and S. Easterbrook. Communication Problems in Requirements Engineering: A Field Study. In *Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering*. Royal Society, London, February 1996.

[AS02] Ian Alexander and Richard Stevens. *Writing Better Requirements*. Pearson Education, 2002.

[ATB06] Thomas A. Alspaugh, Bill Tomlinson, and Eric Baumer. Using social agents to visualize software scenarios. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis'06, pages 87–94, New York, NY, USA, 2006. ACM.

[Avc08] Oral Avci. Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre. *Industrialisierung des Software-Managements*, 139:89–104, 2008. In German.

[BBLZ96] Dirk Bäumer, Walter R. Bischofberger, Horst Lichter, and Heinz Züllighoven. User interface prototyping—concepts, tools, and experience. In *Proc. of the 18th International Conference on Software Engineering*, ICSE'96, pages 532–541, Washington, DC, USA, 1996. IEEE Computer Society.

[BCK+07] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: lessons from customer engagements. *IBM Syst. J.*, 46(4):703–721, 2007.

[BDML09] Robin Bergenthum, Jörg Desel, Sebastian Mauser, and Robert Lorenz. Construction of process models from example runs. In Kurt Jensen and Wil M. Aalst, editors, *Transactions on Petri Nets and Other Models of Concurrency II*, pages 243–259. Springer-Verlag, Berlin, Heidelberg, 2009.

[Ber95] Daniel M. Berry. The importance of ignorance in requirements engineering. *Journal of Systems and Software*, 28(1):179–184, February 1995.

[Ber02] Daniel M. Berry. Controversy corner: the importance of ignorance in requirements engineering: An earlier sighting and a revisitation. *J. Syst. Softw.*, 60(1):83–85, January 2002.

[BH02] Luciano Baresi and Reiko Heckel. Tutorial introduction to graph transformation: A software engineering perspective. In Andrea Corradini, Hartmut Ehrig, Hans Kreowski, and Grzegorz Rozenberg, editors, *Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 402–429. Springer Berlin / Heidelberg, 2002.

[BHK04] Marc Born, Eckhardt Holz, and Olaf Kath. *Softwareentwicklung mit UML 2.* Addison Wesley, 2004.

[BP84] Victor R. Basili and Barry T. Perricone. Software errors and complexity: an empirical investigation. *Commun. ACM*, 27(1):42–52, January 1984.

[BPKR09] Brian Berenbach, Daniel Paulish, Juergen Kazmeier, and Arnold Rudorfer. *Software & Systems Requirements Engineering: In Practice.* McGraw-Hill, Inc., New York, NY, USA, 2009.

[Bro09] Tim Brown. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation.* HarperBusiness, September 2009.

[BSK10] Olesia Brill, Kurt Schneider, and Eric Knauss. Videos vs. use cases: Can videos capture more requirements under time pressure? In Roel Wieringa and Anne Persson, editors, *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *LNCS*, pages 30–44. Springer Berlin Heidelberg, 2010.

[Bux07] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design.* Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2007.

[CA09] Betty H.C. Cheng and Joanne M. Atlee. Current and Future Research Directions in Requirements Engineering. In Kalle Lyytinen, Pericles Loucopoulos, John Mylopoulos, and Bill Robinson, editors, *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *Lecture Notes in Business Information Processing*, pages 11–43. Springer Berlin Heidelberg, 2009.

[Car95] John M. Carroll. Introduction: The scenario perspective on system development. In *Scenario-Based Design: Envisioning Work and Technology in System Development*, chapter Introduction: The Scenario Perspective on System Development, pages 1–18. John Wiley & Sons, Inc., 1995.

[CDL07] Fabio Calefato, Daniela Damian, and Filippo Lanubile. An Empirical Investigation on Text-Based Communication in Distributed Requirements Workshops. In *Proc. of the International Conference on Global Software Engineering*, ICGSE'07, pages 3–11, Washington, DC, USA, 2007. IEEE Computer Society.

[CHP01] Stephen Cranefield, Stefan Haustein, and Martin Purvis. Uml-based ontology modelling for software agents. In *Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents*, 2001.

[CLGG09] John R. Cooper, Seok-Won Lee, Robin A. Gandhi, and Orlena Gotel. Requirements Engineering Visualization: A Survey on the State-of-the-Art. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*, pages 46–55, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph Processes. *Fundam. Inf.*, 26(3-4):241–265, June 1996.

[Coc01] Alistair Cockburn. *Writing Effective Use Cases*. Agile Software Development. Addison Wesley, 2001.

[CP99] Stephen Cranefield and Martin Purvis. Uml as an ontology modelling language. In *In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 46–53, 1999.

[Cro06] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). http://tools.ietf.org/html/rfc4627 (accessed June 2013), July 2006. Request for Comment Nr. 4627.

[CS08] Kevin Clark and Ron Smith. Unleashing the power of design thinking. *Design Management Review*, 19(3):8–15, 2008.

[Dam01] Daniela Damian. An Empirical Study of Requirements Engineering in Distributed Software Projects: Is Distance Negotiation More Effective? In *Proc. of the Asia-Pacific Software Engineering Conference*, page 149, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[Dam07] Daniela Damian. Stakeholders in global requirements engineering: Lessons learned from practice. *IEEE Software*, 24(2):21–27, 2007.

[Dav93] Alan M. Davis. *Software Requirements: Objects, Functions and States*. Prentice-Hall, Englewood Cliffs, revised edition, 1993.

[Dav94] Alan M. Davis. Requirements Engineering. In *Encyclopedia of Software Engineering*, volume II, pages 1043–1055. John Wiley & Sons Inc., February 1994.

[Dav05] Alan M. Davis. *Just Enough Requirements Management : Where Software Development Meets Marketing*. Dorset House Publishing Co., Inc., New York, NY, USA, 2005.

[DDH+06] Alan M. Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proc. of the 14th IEEE International Conference Requirements Engineering*, pages 179–188, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[Des08] Jörg Desel. From human knowledge to process models. In Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, Clemens Szyperski, Roland Kaschek, Christian Kop, Claudia Steinberger, and Günther Fliedl, editors, *Information Systems and e-Business Technologies*, volume 5 of *Lecture Notes in Business Information Processing*, pages 84–95. Springer Berlin Heidelberg, 2008.

[DESG00] Daniela E. Herlea Damian, Armin Eberlein, Mildred L.G. Shaw, and Brian R. Gaines. Using Different Communication Media in Requirements Negotiation. *IEEE Software*, 17(3):28–36, 2000.

[DEVG08] Jonathan Dinerstein, Parris K. Egbert, Dan Ventura, and Michael Goodrich. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence*, 24(4):235–256, November 2008.

[DGZ04] I. Diethelm, L. Geiger, and A. Zündorf. Systematic story driven modeling. In *Proc. of the ICSE Workshop on Scenarios and State Machines: models, algorithms, and tools*, Edinburgh, 2004.

[DHD+07] Alan M. Davis, Ann Hickey, Oscar Dieste, Natalia Juristo, and Ana M. Moreno. A quantitative assessment of requirements engineering publications-1963-2006. In *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, REFSQ'07, pages 129–143, Berlin, Heidelberg, 2007. Springer-Verlag.

[DHP+12] M. von Detten, C. Heinzemann, M. Platenius, J. Rieke, D. Travkin, and S. Hildebrandt. Story diagrams - syntax and semantics. Technical Report tr-ri-12-324, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, July 2012.

[DMCS05] Nirmit Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31:1015–1027, 2005.

[DN07] Alan M. Davis and Kesav V. Nori. Requirements, plato's cave, and perceptions of reality. In *Annual International Computer Software and Applications Conference*, COMPSAC'07, pages 487–492, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[DP06] Brian Dobing and Jeffrey Parsons. How uml is used. *Commun. ACM*, 49(5):109–113, 2006.

[DQS10] Teduh Dirgahayu, Dick Quartel, and Marten van Sinderen. Interaction refinement in the design of business collaborations. In *Proc. of the 2010 ACM Symposium on Applied Computing*, SAC'10, pages 86–93, New York, NY, USA, 2010. ACM.

[DS99] Linda Dawson and Paul Swatman. The use of object-oriented models in requirements engineering: a field study. In *ICIS'99: Proceedings of the 20th international conference on Information Systems*, pages 260–273, Atlanta, GA, USA, 1999. Association for Information Systems.

[EB04] Claudia Ermel and Roswitha Bardohl. Scenario animation for visual behavior models: A generic approach. *Software and Systems Modeling*, 3(2):164–177, 2004.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS. Springer-Verlag Berlin Heidelberg, 2006.

[EHKZ05] Claudia Ermel, Karsten Hölscher, Sabine Kuske, and Paul Ziemann. Animated simulation of integrated uml behavioral models based on graph transformation. In *Proc. of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 125–133, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[Eic12] Daniel Eichler. Empowering stakeholders to manipulate formal models using structured natural language representation. Master's thesis, Hasso Plattner Institute for Software Systems Engineering, University of Potsdam, Germany, March 2012.

[ER03] Albert Endres and Dieter Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. The Fraunhofer IESE series on software engineering. Addison Wesley, 2003.

[FNTZ00] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language and java. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 157–167. Springer Berlin / Heidelberg, 2000.

[Gab09] Gregor Gabrysiak. Modeling and Simulation of Reusable Collaborations for Embedded Systems with Dynamic Structures. Master's thesis, Hasso Plattner Institute for Software Systems Engineering, University of Potsdam, Germany, March 2009.

[Gab11] Gregor Gabrysiak. Exploration and validation through animation of scenario specifications. In *Doctoral Symposium of the 19th IEEE International Requirements Engineering Conference*, RE'11, pages 27–30, Trento, Italy, August 29 2011.

[Gav91] William W. Gaver. Technology affordances. In *CHI'91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 79–84, New York, NY, USA, 1991. ACM.

[GC04] Kentaro Go and John M. Carroll. The blind men and the elephant: views of scenario-based system design. *Interactions*, 11:44–53, November 2004.

[GEGS10] Gregor Gabrysiak, Jonathan A. Edelman, Holger Giese, and Andreas Seibel. How tangible can virtual prototypes be? In *Proc. of the 8th Design Thinking Research Symposium*, DTRS'10, pages 163–174, Sydney, Australia, October 19-20 2010.

[GEHG13] Gregor Gabrysiak, Daniel Eichler, Regina Hebig, and Holger Giese. Enabling Domain Experts to Modify Formal Models via a Natural Language Representation Consistently. In *Proc. of the First ICSE 2013 Workshop on Natural Language Analysis in Software Engineering*, NaturaLiSE'13, pages 1–8, San Francisco, CA, USA, May 25 2013.

[Gem04] Andrew Gemino. Empirical comparisons of animation and narration in requirements validation. *Requir. Eng.*, 9(3):153–168, 2004.

[GGB12] Gregor Gabrysiak, Holger Giese, and Thomas Beyhl. Virtual Multi-User Software Prototypes III. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research – Measuring Performance in Context*, Understanding Innovation, pages 263–284. Springer Berlin Heidelberg, 2012.

[GGHG12] Gregor Gabrysiak, Markus Guentert, Regina Hebig, and Holger Giese. Teaching Requirements Engineering with Authentic Stakeholders: Towards

a Scalable Course Setting. In *Proc. of the First International Workshop on Software Engineering Education Based on Real-Word Experiences*, EduRex'12, pages 1–4, Zurich, Switzerland, June 9 2012.

[GGLS11] Gregor Gabrysiak, Holger Giese, Alexander Lüders, and Andreas Seibel. How Can Metamodels Be Used Flexibly? In *Proc. of ICSE 2011 Workshop on Flexible Modeling Tools*, FlexiTools'11, Waikiki, Honolulu, Hawaii, USA, May 22 2011.

[GGS09] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*, REV'09, pages 41–45, Los Alamitos, CA, USA, September 1 2009.

[GGS10a] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Deriving behavior of multi-user processes from interactive requirements validation. In *Proc. of the IEEE/ACM International Conference on Automated Software Engineering*, ASE'10, pages 355–356, Antwerp, Belgium, September 20-24 2010.

[GGS10b] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Using Ontologies for Flexibly Specifying Multi-User Processes. In *Proc. of ICSE 2010 Workshop on Flexible Modeling Tools*, FlexiTools'10, Cape Town, South Africa, May 2 2010.

[GGS11a] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Towards Next Generation Design Thinking: Scenario-Based Prototyping for Designing Complex Software Systems with Multiple Users. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking: Understand – Improve – Apply*, Understanding Innovation, pages 219–236. Springer Berlin Heidelberg, 2011.

[GGS11b] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Why Should I Help You to Teach Requirements Engineering? In *Proc. of the 6th International Workshop on Requirements Engineering Education and Training*, REET'11, pages 9–13, Trento, Italy, August 29 2011.

[GGS12] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Towards Next-Generation Design Thinking II: Virtual Multi-User Software Prototypes. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research*, Understanding Innovation, pages 107–126. Springer Berlin Heidelberg, 2012.

[GGSN10] Gregor Gabrysiak, Holger Giese, Andreas Seibel, and Stefan Neumann. Teaching requirements engineering with virtual stakeholders without software engineering knowledge. In *Proc. of the 5th International Workshop on*

Requirements Engineering Education and Training, REET'10, pages 36 – 45, Sydney, Australia, September 28 2010.

[GGZ+05] Lars Grunske, Leif Geiger, Albert Zündorf, Niels Eetvelde, Pieter Gorp, and Dániel Varró. Using graph transformation for practical model-driven software engineering. In Sami Beydeda, Matthias Book, and Volker Gruhn, editors, *Model-Driven Software Development*, pages 91–117. Springer Berlin Heidelberg, 2005.

[GH06] Paul Guyot and Shinichi Honiden. Agent-based participatory simulations: Merging multi-agent systems and role-playing games. *Journal of Artificial Societies and Social Simulation*, 9(4), 2006.

[GHG12a] Gregor Gabrysiak, Regina Hebig, and Holger Giese. Decoupled Model-Based Elicitation of Stakeholder Scenarios. In *Proc. of the Seventh International Conference on Software Engineering Advances*, ICSEA'12, pages 70–77, Lisbon, Portugal, November 18-23 2012.

[GHG12b] Gregor Gabrysiak, Regina Hebig, and Holger Giese. Simulation-Assisted Elicitation and Validation of Behavioral Specifications for Multiple Stakeholders. In *Proc. of the 21st IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE'12, pages 220–225, Toulouse, France, June 25-27 2012.

[GHPG13] Gregor Gabrysiak, Regina Hebig, Lukas Pirl, and Holger Giese. Cooperating with a Non-governmental Organization to Teach Gathering and Implementation of Requirements. In *Proc. of the 26th Conference on Software Engineering Education and Training*, CSEE&T'13, pages 11–20, San Francisco, CA, USA, May 19-21 2013.

[GHS09] Holger Giese, Stephan Hildebrandt, and Andreas Seibel. Improved Flexibility and Scalability by Interpreting Story Diagrams. In Tiziana Magaria, J. Padberg, and Gabriele Taentzer, editors, *Proc. of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)*, 2009.

[Gli00] Martin Glinz. Improving the quality of requirements with scenarios. In *Proc. of the 2nd World Congress for Software Quality*, pages 55–60, Yokohama, September 2000.

[GMM07] Orlena C.Z. Gotel, Francis T. Marchese, and Stephen J. Morris. On requirements visualization. In *Proc. of the 2nd International Workshop on Requirements Engineering Visualization*, page 11, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[GMM09] Carlo Ghezzi, Andrea Mocci, and Mattia Monga. Synthesizing intensional behavior models by graph transformation. In *ICSE'09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 430–440, Washington, DC, USA, 2009. IEEE Computer Society.

[Gre76] Anthony G. Greenwald. Within-Subjects Designs: To Use or Not To Use? *Psychological Bulletin*, 83(2):314 – 320, 1976.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. Knowledge Acquisition 5(2):199-220, Stanford University, April 1993.

[GW07] Martin Glinz and Roel J. Wieringa. Guest editors' introduction: Stakeholders in requirements engineering. *IEEE Software*, 24(2):18–20, 2007.

[GZ04] Leif Geiger and Albert Zündorf. Statechart modeling with fujaba. In *Proc. of the International Workshop on Graph-Based Tools*, volume Electronic Notes in Theoretical Computer Science 127 of *GraBaTs'04*, pages 37–49, 2004.

[GZ05] Vincenzo Gervasi and Didar Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Trans. Softw. Eng. Methodol.*, 14(3):277–330, 2005.

[Ham94] Richard Hamlet. Random Testing. In *Encyclopedia of Software Engineering*, volume II, pages 971–978. John Wiley & Sons Inc., February 1994.

[Har01] David Harel. From play-in scenarios to code: An achievable dream. *Computer*, 34:53–60, January 2001.

[HBJ09] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. Cope – automating coupled evolution of metamodels and models. In Sophia Drossopoulou, editor, *ECOOP 2009 – Object-Oriented Programming*, volume 5653 of *LNCS*, pages 52–76. Springer Berlin / Heidelberg, 2009.

[HD98] Patrick Heymans and Eric Dubois. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. *Requirements Engineering*, 3(3-4):202–218, March 1998.

[Hec06] Reiko Heckel. Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187 – 198, 2006. Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004).

[HHT96] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3):287–313, 1996.

[HJD11] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering*. Springer, 3rd edition, 2011.

[HKMP02] David Harel, Hillel Kugler, Rami Marelly, and Amir Pnueli. Smart play-out of behavioral requirements. In *FMCAD'02: Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, pages 378–398, London, UK, 2002. Springer-Verlag.

[HKT02] Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In Andrea Corradini, Hartmut Ehrig, Hans-Jrg Kreowski, and Grzegorz Rozenberg, editors, *Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 161–176. Springer Berlin Heidelberg, 2002.

[HM03a] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[HM03b] David Harel and Rami Marelly. Specifying and executing behavioral requirements: the play-in/play-out approach. *Software and Systems Modeling*, 2:82–107, 2003. 10.1007/s10270-002-0015-5.

[HPW98] Peter Haumer, Klaus Pohl, and Klaus Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering*, 24(12):1036–1054, 1998.

[HSKS08] David Harel, Itai Segall, Hillel Kugler, and Yaki Setty. Crafting game-models using reactive system design. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Future Play'08, pages 121–128, New York, NY, USA, 2008. ACM.

[IEE90] IEEE Standards Board. *IEEE Standard Glossary of Software Engineering Terminology (610.12-1990)*. The Institute of Electrical and Electronics Engineers, New York, 1990. Reaffirmed 2002.

[IEE98] IEEE. Recommended practice for software requirements specifications. Technical Report IEEE Std. 830-1998, IEEE, 1998.

[Jac00] Michael Jackson. The real world. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of C A R Hoare*, pages 157–173, 2000.

[JMF09] Ivan Jureta, John Mylopoulos, and Stephane Faulkner. Analysis of multi-party agreement in requirements validation. In *Proc. of the 17th IEEE International Requirements Engineering Conference (RE'09)*, pages 57–66, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[JP93] M. Jarke and K. Pohl. Establishing visions in context – towards a model of requirements processes. In J. I. DeGross, R. P. Ostrom, and D. Robey, editors, *Proceedings of the 14th Intl. Conference on Information Systems*, pages 23–34, Orlando, Florida, USA, Dezember 1993.

[KCH+02] Paul Kogut, Stephen Cranefield, Lewis Hart, Mark Dutra, Kenneth Baclawski, Mieczyslaw Kokar, and Jeffrey Smith. Uml for ontology development. *Knowl. Eng. Rev.*, 17(1):61–64, March 2002.

[KG00] Daryl Kulak and Eamonn Guiney. *Use Cases: Requirements in Context.* Addison Wesley, 1st edition, May 2000.

[Kle11] Stefan Kleff. Effiziente Simulation von virtuellen Prototypen. Master's thesis, Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, Germany, August 2011.

[KNNZ00] Hans J. Köhler, Ulrich Nickel, Jörg Niere, and Albert Zündorf. Integrating uml diagrams for production control systems. In *Proc. of the 22nd International Conference on Software Engineering*, ICSE'00, pages 241–251, New York, NY, USA, 2000. ACM.

[Küh06] Thomas Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5:369–385, 2006.

[Kuu95] K. Kuutti. Work processes: Scenarios as a preliminary vocabulary. In John M. Carroll, editor, *Scenario-Based Design: Envisioning Work and Technology in System Development*, pages 19–36. John Wiley, 1995.

[Lam01] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. of the 5th International Symposium on Requirements Engineering (RE'01)*, page 0249, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[Lau02] Soren Lauesen. *Software Requirements: Styles and Techniques.* Addison-Wesley, Longman, Amsterdam, 2002.

[LCL87] F. J. Lin, P. M. Chu, and M. T. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *SIGCOMM Comput. Commun. Rev.*, 17(5):126–135, August 1987.

[LDD06] Hongzhi Liang, Juergen Dingel, and Zinovy Diskin. A comparative survey of scenario-based to state-based model synthesis approaches. In *Proc. of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, SCESM'06, pages 5–12, New York, NY, USA, 2006. ACM.

[Lef11] Dean Leffingwell. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Agile Software Development. Addison Wesley, 2011.

[LK10] Alfons Laarman and Ivan Kurtev. Ontological metamodeling with explicit instantiation. In Mark van den Brand, Dragan Gaševic, and Jeff Gray, editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 174–183. Springer Berlin / Heidelberg, 2010.

[LL93] V. Lalioti and P. Loucopoulos. Visualisation for validation. In Colette Rolland, Franois Bodart, and Corine Cauvet, editors, *Advanced Information Systems Engineering*, volume 685 of *Lecture Notes in Computer Science*, pages 143–164. Springer Berlin / Heidelberg, 1993.

[LMP04] Mich Luisa, Franch Mariangela, and NoviInverardi Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9:40–56, 2004. 10.1007/s00766-003-0179-8.

[LRA02] Wesley James Lloyd, Mary Beth Rosson, and James D. Arthur. Effectiveness of Elicitation Techniques in Distributed Requirements Engineering. In *Proc. of the 10th Anniversary Joint IEEE International Requirements Engineering Conference (RE'02)*, page 311, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[LS87] Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, 1987.

[LST08] Youn-Kyung Lim, Erik Stolterman, and Josh Tenenberg. The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Trans. Comput.-Hum. Interact.*, 15(2):1–27, 2008.

[LSW08] Daniel Lübke, Kurt Schneider, and Matthias Weidlich. Visualizing use case sets as bpmn processes. In *Proc. of the 3rd International Workshop on Requirements Engineering Visualization (REV'08)*, pages 21–25, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[Lue11] A. Luebbe. *Tangible Business Process Modeling: Design and Evaluation of a Process Model Elicitation Technique*. PhD thesis, University of Potsdam, 2011.

[LW99] Dean Leffingwell and Don Widrig. *Managing Software Requirements*. Addison Wesley, 1999.

[LW11] A. Luebbe and M. Weske. Tangible media in process modeling – a controlled experiment. In H. Mouratidis and C. Roland, editors, *Proc. of the 23th Conference on Advanced Information Systems Engineering (CAiSE 2011)*, volume 6741 of *LNCS*, pages 283–298, 2011.

[MAA08] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating natural language specifications from uml class diagrams. *Requir. Eng.*, 13(1):1–18, 2008.

[Mai12] Neil Maiden. Cherishing ambiguity. *IEEE Software*, 29(6):16–17, 2012.

[MDF05] Geoffrey Marczyk, David DeMatteo, and David Festinger. *Essentials of Research Design and Methodology*. Essentials of Behavioral Science. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[Mer13] Merriam-Webster Online Dictionary. Definition of *Black Box*. http://www.merriam-webster.com/dictionary/black-box, Accessed June 2013.

[MNK+07] Neil Maiden, Cornelius Ncube, Simin Kamali, Norbert Seyff, and Paul Grünbacher. Exploring scenario forms and ways of use to discover requirements on airports that minimize environmental impact. In *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE'07)*, pages 29–38, 2007.

[MPGK00] Jeff Magee, Nat Pryce, Dimitra Giannakopoulou, and Jeff Kramer. Graphical animation of behavior models. In *Proc. of the 22nd International Conference on Software Engineering*, ICSE'00, page 499, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[Mur11] Fernando Muradas. A novel framework for requirements elicitation in a military settings. In *Doctoral Symposium of the 19th IEEE International Requirements Engineering Conference*, RE'11, Trento, Italy, August 29 2011.

[NC03] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42(3):428–445, 2003.

[NL03] Colin J. Neill and Phillip A. Laplante. Requirements Engineering: The State of the Practice. *IEEE Software*, 20(6):40–45, 2003.

[NNZ00] Ulrich Nickel, Jörg Niere, and Albert Zündorf. The fujaba environment. In *Proc. of the 22nd international conference on Software engineering*, ICSE'00, pages 742–745, New York, NY, USA, 2000. ACM.

[NS00] L. Nguyen and P. A. Swatman. Essential and incidental complexity in requirements models. In *Proc. of the 4th International Conference on Requirements Engineering (RE'00)*, page 130, Washington, DC, USA, 2000. IEEE Computer Society.

[OBS+10] Harold Ossher, Rachel Bellamy, Ian Simmonds, David Amid, Ateret Anaby-Tavor, Matthew Callery, Michael Desmond, Jacqueline de Vries, Amit Fisher, and Sophia Krasikov. Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, OOPSLA'10, pages 848–864, New York, NY, USA, 2010. ACM.

[OHS+10] Harold Ossher, André van der Hoek, Margaret-Anne Storey, John Grundy, and Rachel Bellamy. Flexible modeling tools (flexitools2010). In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE'10, pages 441–442, New York, NY, USA, 2010. ACM.

[OMG05a] Object Management Group. Unified Modeling Language: Infrastructure. http://www.omg.org/spec/UML/2.0/ (accessed June 2013), July 2005. Version 2.0.

[OMG05b] Object Management Group. Unified Modeling Language: Superstructure. http://www.omg.org/spec/UML/2.0/ (accessed June 2013), July 2005. Version 2.0.

[OMG05c] Object Management Group. XML Metadata Interchange. http://www.omg.org/spec/XMI/2.1/ (accessed June 2013), September 2005. Version 2.1.

[OP10] Alexander Osterwalder and Yves Pigneur. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. John Wiley & Sons, Inc., August 2010.

[ÖPMS98] M. Özcan, P. Parry, I. Morrey, and J. Siddiqi. Visualisation of executable formal specifications for user validation. In Tiziana Margaria, Bernhard Steffen, Roland Rückert, and Joachim Posegga, editors, *Services and Visualization Towards User-Friendly Design*, volume 1385 of *Lecture Notes in Computer Science*, pages 142–157. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0053503.

[PBM+05] Christophe Ponsard, Nadiya Balych, Philippe Massonet, Jean Vanderdonckt, and Axel van Lamsweerde. Goal-Oriented Design of Domain Control Panels. In Stephen W. Gilroy and Michael D. Harrison, editors, *DSV-IS*, volume 3941 of *LNCS*, pages 249–260. Springer, 2005.

[PMW09] Hasso Plattner, Christoph Meinel, and Ulrich Weinberg. *Design Thinking*. mi-Wirtschaftsbuch, 2009. In German.

[Poe99] Stephan Poelmans. Workarounds and distributed viscosity in a workflow system: a case study. *SIGGROUP Bull.*, 20(3):11–12, 1999.

[Poh93] Klaus Pohl. The Three Dimensions of Requirements Engineering. In *CAiSE'93: Proceedings of Advanced Information Systems Engineering*, pages 275–292, London, UK, 1993. Springer-Verlag.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[Pow10] Daniel Powell. Behavior engineering - a scalable modeling and analysis method. In *Proc. of the 8th IEEE International Conference on Software Engineering and Formal Methods*, pages 31–40, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[PR11] Klaus Pohl and Chris Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level*. Rocky Nook, 2011.

[Pre05] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 6th edition, 2005.

[Ric11] Stefan Richter. Gesteuerte und interaktive Simulation von virtuellen Prototypen. Master's thesis, Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, Germany, November 2011.

[Rot89] Jeff Rothenberg. The nature of modeling. In Lawrence E. Widman, Kenneth A. Loparo, and Norman R. Nielsen, editors, *AI, Simulation & Modeling*, pages pp. 75–92. John Wiley & Sons, Inc., 1989.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1 of *Foundations*. World Scientific Pub. Co., 1997.

[RS09] Chris Rupp and die SOPHISTen. *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*. Carl Hanser Verlag GmbH & CO. KG, 5th edition, July 2009.

[Sar05] Edward P. Sarafino. *Research Methods: Using Processes & Procedures of Science to Understand Behavior*. Upper Saddle River: Pearson/Prentice Hall, 2005.

[SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. edition, 2009.

[Sch96] Kurt Schneider. Prototypes as assets, not toys: why and how to extract knowledge from prototypes. In *Proc. of the 18th International Conference on Software Engineering*, ICSE'96, pages 522–531, Washington, DC, USA, 1996. IEEE Computer Society.

[Sch04] Michael Schrage. Never Go to a Client Meeting without a Prototype. *IEEE Software*, 21:42–45, 2004.

[Sch07] Kurt Schneider. Generating fast feedback in requirements elicitation. In Pete Sawyer, Barbara Paech, and Patrick Heymans, editors, *Requirements Engineering: Foundation for Software Quality*, volume 4542 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin Heidelberg, 2007.

[SGM05] Christian Seybold, Martin Glinz, and Silvio Meier. Simulation-based Validation and Defect Localization for Evolving, Semi-Formal Requirements Models. In *Proc. of the Asia-Pacific Software Engineering Conference*, pages 408–420, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[SL04] Keng Siau and Lihyunn Lee. Are use case and class diagrams complementary in requirements analysis? an experimental study on use case and class diagrams in uml. *Requirements Engineering*, 9:229–237, 2004.

[SMG04] Christian Seybold, Silvio Meier, and Martin Glinz. Evolution of requirements models by simulation. In *Proc. of the International Workshop on Principles of Software Evolution*, pages 43–48, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[SMG06] Christian Seybold, Silvio Meier, and Martin Glinz. Scenario-driven modeling and validation of requirements models. In *Proc. of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, SCESM'06, pages 83–89, New York, NY, USA, 2006. ACM.

[SMK+09] Norbert Seyff, Neil Maiden, Kristine Karlsen, James Lockerbie, Paul Grünbacher, Florian Graf, and Cornelius Ncube. Exploring how to use scenarios to discover requirements. *Requirements Engineering*, 14(2):91–111, June 2009.

[SML+09] Katherine M. Sellen, Micheal A. Massimi, Danielle M. Lottridge, Khai N. Truong, and Sean A. Bittle. The people-prototype problem: understanding the interaction between prototype format and user group. In *CHI'09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 635–638, New York, NY, USA, 2009. ACM.

[Sny03] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann, 2003.

[Som06] Ian Sommerville. *Software Engineering*. Addison Wesley, 8th edition, 2006.

[SRB+00] Reto Schmid, Johannes Ryser, Stefan Berner, Martin Glinz, Ralf Reutemann, and Erwin Fahr. A Survey of Simulation Tools for Requirements Engineering. Technical report, University of Zurich, 2000.

[SS97] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[SS99] D. Skog and M. Söderlund. Virtual information representation. In *Proceedings of the 22nd Information Systems Research Seminar in Scandinavia*, pages 191–204, Keuruu, Finland, 1999.

[SS12a] Kai Stapel and Kurt Schneider. FLOW-Methode - Methodenbeschreibung zur Anwendung von FLOW. Technical report, Fachgebiet Software Engineering, Leibniz Universität Hannover, 2012.

[SS12b] Marc Stickdorn and Jakob Schneider, editors. *This is Service Design Thinking: Basics – Tools – Cases*. Bis Publishers, 2012.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie (German)*. Springer, Wien, 1973.

[Teu11] Ralf Teusner. Smarte Simulation von virtuellen Prototypen. Master's thesis, Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, Germany, November 2011.

[Tey02] A. Teyseyre. A 3d visualization approach to validate requirements. In *Proc. of the Congreso Argentino de Ciencias de la Computación*, Argentina, October 2002.

[TGRK13] Ralf Teusner, Gregor Gabrysiak, Stefan Richter, and Stefan Kleff. Interactive Strategy-Based Validation of Behavioral Models. In *Proc. of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques*, GT-VMT'13, Rome, Italy, March 23-24 2013.

[UCKM04] Sebastian Uchitel, Robert Chatley, Jeff Kramer, and Jeff Magee. Fluent-based animation: Exploiting the relation between goals and scenarios for requirements validation. In *Proc. of the 12th IEEE International Conference on Requirements Engineering (RE'04)*, pages 208–217, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[UKM01] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Detecting implied scenarios in message sequence chart specifications. *SIGSOFT Softw. Eng. Notes*, 26:74–82, September 2001.

[UKM02a] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Implied scenario detection in the presence of behaviour constraints. *Electronic Notes in Theoretical Computer Science*, 65(7):65 – 84, 2002. VISS 2002, Validation and Implementation of Scenario-based Specifications (Satellite Event of ETAPS 2002).

[UKM02b] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Negative scenarios for implied scenario elicitation. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT'02/FSE-10, pages 109–118, New York, NY, USA, 2002. ACM.

[Usc98] Mike Uschold. Knowledge level modelling: concepts and terminology. *Knowl. Eng. Rev.*, 13(1):5–29, 1998.

[Val91] Antti Valmari. Stubborn sets for reduced state generation. In *Proceedings on Advances in Petri nets 1990*, pages 491–515, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[VBI+09] Eurico Vasconcelos, Jean-Pierre Briot, Marta Irving, Simone Barbosa, and Vasco Furtado. A user interface to support dialogue and negotiation in participatory simulations. In Nuno David and Jaime Sichman, editors, *Multi-Agent-Based Simulation IX*, volume 5269 of *Lecture Notes in Computer Science*, pages 127–140. Springer Berlin / Heidelberg, 2009.

[VLMP04] Hung Tran Van, Axel van Lamsweerde, Philippe Massonet, and Christophe Ponsard. Goal-oriented requirements animation. In *Proc. of the IEEE International Conference on Requirements Engineering*, volume 0, pages 218–228, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[WJ10] Jon Whittle and Praveen K. Jayaraman. Synthesizing hierarchical state machines from expressive scenario descriptions. *ACM Trans. Softw. Eng. Methodol.*, 19:8:1–8:45, February 2010.

[WRH+00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[ZBCG10] William G. Zikmund, Barry J. Babin, Jon C. Carr, and Mitch Griffin. *Business Research Methods*. Wadsworth Inc Fulfillment, 8th edition, 2010.

[ZD09] Albert Zündorf and Jörn Dreyer. NT2OD – from natural text to object diagram. In *Proc. of the 7th international Fujaba Days*, pages 56–58, November 2009.

[ZG03] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993 – 1009, 2003. Eighth International Workshop on Requirements Engineering: Foundation for Software Quality.

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6:1–30, January 1997.

[Zün01] Albert Zündorf. Rigorous Object Oriented Software Development, 2001. Habilitation Thesis, University of Paderborn.

A. Publications

Our initial idea of including stakeholders interactively into the validation of requirements was first published at the 4^{th} *International Workshop on Requirements Engineering Visualization* (REV'09, [GGS09]). It was motivated through the needs which we elicited from the design consultancy D-LABS GmbH,[1] an industrial partner within our HPDTRP[2] project, and which were published in the book *Design Thinking: Understand – Improve – Apply* [GGS11a]. As the idea of model-based prototyping aimed at stakeholders manifested, we also discussed the similarities that exist between models and prototypes at the 8^{th} *Design Thinking Research Symposium* (DTRS8, [GEGS10]). Then, the interactivity of our approach was evaluated in an experiment with students and published in the book *Design Thinking Research: Studying Co-Creation in Practice* [GGS12]. Also, we implemented and presented the capability to derive formal specifications based on observed interactions at the 25^{th} *IEEE/ACM International Conference on Automated Software Engineering* (ASE'10, [GGS10a]). The overall dissertation topic was then presented and discussed at the *Doctoral Symposium* that was part of the 19^{th} *IEEE International Requirements Engineering Conference* (RE'11, [Gab11]). Additionally, the three years of our HPDTRP project were summarized along with experimental results concerning the understandability of our approach in the book *Design Thinking Research: Measuring Performance in Context* [GGB12]. Further, the simulation loop was established and subsequently published at the 21^{st} *IEEE International Conference on Collaboration Technologies and Infrastructures* (CoMetS'12, [GHG12b]). Then, an extension which defined a black box abstraction for stakeholders was introduced and published at the 7^{th} *International Conference on Software Engineering Advances* (ICSEA'12, [GHG12a]).

Furthermore, we published papers describing the results of master's theses building upon and extending the concepts developed within this thesis at the 12^{th} *International Workshop on Graph Transformation and Visual Modeling Techniques* (GT-VMT'13, [TGRK13]) and at the 1^{st} *International Workshop on Natural Language Analysis in Software Engineering* (NaturaLiSE'13, [GEHG13]).

Along the way, two related topics emerged and were looked into as well. Firstly, we investigated the potentials of flexible metamodels, i.e. how our approach can benefit from them (*FlexiTools Workshop at ICSE'10* [GGS10b]) and how metamodels can be employed flexibly in general (*FlexiTools Workshop at ICSE'11* [GGLS11]). Secondly, based on the need for suitable experimentees, we came up with a novel approach of recruiting

[1] `http://www.d-labs.com/english/` (accessed June 2013)

[2] HPI-Stanford Design Thinking Research Program, `http://www.hpi.uni-potsdam.de/forschung/design_thinking_research_program/program/?L=1` (accessed June 2013)

and instructing simulated stakeholders (5^{th} *REET Workshop at RE'10* [GGSN10]). This led to a new concept for a requirements engineering seminar (6^{th} *REET Workshop at RE'11* [GGS11b]) which was successfully conducted three times at the HPI. While the second seminar in cooperation with a software startup company was described in more detail focusing on the scalability of the seminar setting (*EduREX Workshop at ICSE'12* [GGHG12]), the results of the first seminar were discussed along with the subsequent implementation in our publication at the 26^{th} *Conference on Software Engineering Education and Training* (CSEE&T'13, [GHPG13]).

Publications

[Gab11] Gregor Gabrysiak. Exploration and validation through animation of scenario specifications. In *Doctoral Symposium of the 19th IEEE International Requirements Engineering Conference*, RE'11, pages 27–30, Trento, Italy, August 29 2011.

[GEGS10] Gregor Gabrysiak, Jonathan A. Edelman, Holger Giese, and Andreas Seibel. How tangible can virtual prototypes be? In *Proc. of the 8th Design Thinking Research Symposium*, DTRS'10, pages 163–174, Sydney, Australia, October 19-20 2010.

[GEHG13] Gregor Gabrysiak, Daniel Eichler, Regina Hebig, and Holger Giese. Enabling Domain Experts to Modify Formal Models via a Natural Language Representation Consistently. In *Proc. of the First ICSE 2013 Workshop on Natural Language Analysis in Software Engineering*, NaturaLiSE'13, pages 1–8, San Francisco, CA, USA, May 25 2013.

[GGB12] Gregor Gabrysiak, Holger Giese, and Thomas Beyhl. Virtual Multi-User Software Prototypes III. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research – Measuring Performance in Context*, Understanding Innovation, pages 263–284. Springer Berlin Heidelberg, 2012.

[GGHG12] Gregor Gabrysiak, Markus Guentert, Regina Hebig, and Holger Giese. Teaching Requirements Engineering with Authentic Stakeholders: Towards a Scalable Course Setting. In *Proc. of the First International Workshop on Software Engineering Education Based on Real-Word Experiences*, EduRex'12, pages 1–4, Zurich, Switzerland, June 9 2012.

[GGLS11] Gregor Gabrysiak, Holger Giese, Alexander Lüders, and Andreas Seibel. How Can Metamodels Be Used Flexibly? In *Proc. of ICSE 2011 Workshop on Flexible Modeling Tools*, FlexiTools'11, Waikiki, Honolulu, Hawaii, USA, May 22 2011.

[GGS09] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*, REV'09, pages 41–45, Los Alamitos, CA, USA, September 1 2009.

[GGS10a] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Deriving behavior of multi-user processes from interactive requirements validation. In *Proc. of the IEEE/ACM International Conference on Automated Software Engineering*, ASE'10, pages 355–356, Antwerp, Belgium, September 20-24 2010.

[GGS10b] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Using Ontologies for Flexibly Specifying Multi-User Processes. In *Proc. of ICSE 2010 Workshop on Flexible Modeling Tools*, FlexiTools'10, Cape Town, South Africa, May 2 2010.

[GGS11a] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Towards Next Generation Design Thinking: Scenario-Based Prototyping for Designing Complex Software Systems with Multiple Users. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking: Understand – Improve – Apply*, Understanding Innovation, pages 219–236. Springer Berlin Heidelberg, 2011.

[GGS11b] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Why Should I Help You to Teach Requirements Engineering? In *Proc. of the 6th International Workshop on Requirements Engineering Education and Training*, REET'11, pages 9–13, Trento, Italy, August 29 2011.

[GGS12] Gregor Gabrysiak, Holger Giese, and Andreas Seibel. Towards Next-Generation Design Thinking II: Virtual Multi-User Software Prototypes. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research*, Understanding Innovation, pages 107–126. Springer Berlin Heidelberg, 2012.

[GGSN10] Gregor Gabrysiak, Holger Giese, Andreas Seibel, and Stefan Neumann. Teaching requirements engineering with virtual stakeholders without software engineering knowledge. In *Proc. of the 5th International Workshop on Requirements Engineering Education and Training*, REET'10, pages 36 – 45, Sydney, Australia, September 28 2010.

[GHG12a] Gregor Gabrysiak, Regina Hebig, and Holger Giese. Decoupled Model-Based Elicitation of Stakeholder Scenarios. In *Proc. of the Seventh International Conference on Software Engineering Advances*, ICSEA'12, pages 70–77, Lisbon, Portugal, November 18-23 2012.

[GHG12b] Gregor Gabrysiak, Regina Hebig, and Holger Giese. Simulation-Assisted Elicitation and Validation of Behavioral Specifications for Multiple Stakeholders. In *Proc. of the 21st IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE'12, pages 220–225, Toulouse, France, June 25-27 2012.

[GHPG13] Gregor Gabrysiak, Regina Hebig, Lukas Pirl, and Holger Giese. Cooperating with a Non-governmental Organization to Teach Gathering and Implementation of Requirements. In *Proc. of the 26th Conference on Software Engineering Education and Training*, CSEE&T'13, pages 11–20, San Francisco, CA, USA, May 19-21 2013.

[TGRK13] Ralf Teusner, Gregor Gabrysiak, Stefan Richter, and Stefan Kleff. Interactive Strategy-Based Validation of Behavioral Models. In *Proc. of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques*, GT-VMT'13, Rome, Italy, March 23-24 2013.

B. Evaluation Data

B.1. T-Test Results for Section 8.2.1

Table B.1 presents the p values of independent two-sample t-tests of the eleven Likert scale questions. As a special form of ANalysis Of VAriance (*ANOVA*), t-tests can be used to determine whether a statistically significant difference between the means of two groups exist [MDF05]. Therefore, apart from the three pairwise t-tests, the last column shows the corresponding ANOVA results which determine significance of the responses of all three groups for each question.

	p for comparing the groups using ...			ANOVA
#	VP / UML	VP / Pen and Paper	UML / Pen and Paper	results
Q_1	0.63332	0.09726	0.05800	0.12
Q_2	0.20676	0.79241	0.36356	0.47
Q_3	1.00000	**0.00487**	**0.01591**	**0.009**
Q_4	0.47453	0.08772	**0.02376**	0.064
Q_5	0.83983	1.00000	0.85786	0.97
Q_6	0.80436	0.80651	0.63332	0.89
Q_7	0.13058	0.11745	0.64402	0.22
Q_8	**0.00968**	0.76396	**0.02844**	**0.019**
Q_9	**0.00145**	**0.01084**	**0.00001**	**0.0001**
Q_{10}	0.74266	0.07977	**0.03653**	0.13
Q_{11}	0.06365	0.70655	0.06801	0.10

Table B.1.: Results of *independent two-sample t-tests* which evaluate pairwise the significance of either of the three methods used by the different groups (*VP* stands for virtual prototype and *UML* for the modeling tool; bold values are considered significant, i.e. ≤ 0.05)

B.2. Summarized Responses of the Evaluation in Section 8.2.2

Subject	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Age	21	25	30	22	28	23	18	23	22	24	21	22	24	22	26
Description for V_1	User entscheidet sich in PC-Spiel für Assi	Auswahl Assi (ausführlich beschrieben)	Auswahl Assistent "man spielt allein"	Auswahl Assistent (PC-Spiel fehlt)	Auswahl der ersten (Assi) Möglichkeit (Simulator)	Auswahl einer Möglichkeit (PC Spiel)	Auswahl einer Rolle	Auswahl einer Rolle	Bürospiel in dem man Rollen einnehmen kann	Simulation Büroablauf, Assi ausgewählt	Wahl in einem Spiel für Assi	Auswahl Assi, ist bereit	Aufforderung erscheint auf Bildschirm und wird bestätigt	Simulationsspiel, Auswahl der Rolle Assi (spielt alleine)	Assi wird ausgewählt (PC spiel)
Confidence	6	6	5	5	6	4	2	5	6	6	5	6	7	7	6
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	**Incorrect**	*Correct*	*Correct*
Description for V_2	Meier bekommt Mail von Boss, ob Anweisung von Schmidt da ist	"Symbol blinkt" frage von Boss, ob Anfrage von Kunde bearbeitet wurde	Mail von Boss an Meier "haben sie Auftrag bekommen"	Assi bekommt Mail von Boss	Mail wird geöffnet (durch klicken auf icon) und gelesen	öffnet Briefumschlag, Fenster öffnet sich und wird geschlossen	Simulation Büroablaufes. Mailprogramm geöffnet	Assi bekommt/liest Mail vom Boss	Post wird gelesen	Assi liest Mail von Boss (anderer Mitspieler)	Mail mit Frage nach Auftrag wird geöffnet	Meier öffnet Mail ob Auftrag da ist	Blinkender Brief, wird gelesen	Spieler ist Meier, Outlook wird geöffnet um Aufträge anzunehmen, er erhält Anfrage	Assi öffnet und liest Mail
Confidence	6	6	5	6	5	5	5	6	–	6	5	7	7	6	5
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	**Incorrect**	*Correct*	**Incorrect**	*Correct*	**Incorrect**	*Correct*	*Correct*
Description for V_3	Anruf vom Boss wegen Kaffee/ in 5 min da	Boss fragt per Telefon nach Kaffee / "in 5min"	Anruf von Boss wegen Kaffee/ in 5min	"Boss telefoniert nach Kaffee/ Maier "ja in 5min"	Telefongespräch wird geführt	Kommunikation per Telefon	Boss telt Assi an will Kaffee/ gleich	Boss ruft Assi an, will Kaffee/ "in 5min"	Telefonat mit Boss	Boss ruft Assi, will Kaffee/ in 5min	Boss ruft Assi an, will Kaffee/ 5min	Telefon, Boss will Kaffee, Meier: "in 5min"	Telefon klingelt, Boss fragt Meier nach Kaffee: "in 5min"	Anruf von Boss, Kaffee kommt in 5min	Boss kontaktiert Assi nach Kaffee, Antwort in 5min
Confidence	7	6	6	5	6	4	–	6	7	6	5	6	7	7	6
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*
Description for V_4	Kunde erstellt Auftragsdokument, auf dem Signum von Boss fehlt	Kunde erstellt Auftragsdokument, verwirft es	Assi erstellt Dokument/Auftrag mit Word. Gegenstand wird verkauft	Assi erstellt word Dokument	Probant erstellt Dokument !lacht!	Auftrag wird erstellt (word)	Benutzer erstellt Autrag Word	Assi erstellt Auftrag und Verwirft, Boss hat sich ausgeloggt	Erstellung eines Dokumentes	Assi erstellt Dokument (Auftrag)	Worderstelltes Dokument "Auftrag"	Jemand erstellt Auftrag/ Dokument	Auftrag/ Dokument wird erstellt	Person ist jetzt Kunde und gibt Auftrag	Assi gibt Auftrag auf
Confidence	7	5	5	3	6	5	5	6	6	6	4	4	5	7	5
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*
Description for V_5	Boss bewilligt und unterschreibt Auftrag	Dokument wird unterschrieben	Boss unterschreibt und speichert Auftrag	Boss erstellt/unterschreibt Auftrag	Probant unterschreibt Auftrag und versendet	Auftrag wird angenommen, unterschrieben und benannt	Boss genehmigt Antrag	Assi, ruft Auftrag auf, unterschreibt ihn an Stelle des Bosses und speichert	Assi unterschreibt Auftrag anstatt Boss	Boss bestätigt Auftrag und unterschreibt	"Assi war aufeinmal der Boss" und nahm Auftrag an	Meier öffnet Auftrag, nimmt an und unterschreibt	Auftrag wird bestätigt und mit Meier unterschrieben	Person ist jetzt Boss, Auftrag wird angenommen, Unterzeichnung mit Meier	Assi unterschreibt Auftrag,"eigenartig, da dies der Boss machen sollte"
Confidence	7	6	5	6	6	5	7	6	5	7	4	3	7	6	6
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*
Description for V_6	Boss verlässt Büro und fragt Assi nach Kaffee/ "in 2min"	klickt auf Tür, fragt Assi nach Kaffee/ "in 2min"	Boss besucht Assi, fragt nach Kaffe/ "in 2 min"	Boss besucht Assi, will Kaffee	Probant klickt auf Tür > redet mit Assi	Tür wird angeklickt, Kommunikation mit Assi	Boss besucht Assi, will Kaffee - "in 2min"	Assi 1 besucht Assi 2 will Kaffee/ "in 2 min"	Kommunikation zw. Boss und Assi	Boss besucht Assi, will Kaffee	Boss fragt Assi nach Kaffee, Antwort in 2min	Boss besucht Assi fragt nach Kaffee	Meier klickt Fenster an fragt Sekretärin nach Kaffee/ in 2min	Boss besucht Assi, fragt nach Kaffee, Antwort: 2min	Meier ist jetzt Boss geworden
Confidence	7	6	5	5	6	4	7	6	6	6	5	6	7	7	6
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	**Incorrect**	*Correct*	**Incorrect**	*Correct*	*Correct*	*Correct*	**Incorrect**
Description for V_7	Kunde schickt Mail an Meier	per "Nachrichtensymbol" Auftrag verschickt	Kunde schickt Assi den Auftrag	Kunde nimmt Auftrag an, schreibt dies an Assi	Probant schreibt Mail, hängt Anhang dran, sendet	Mail benutzt, Nachricht + Anhang verfasst, versendet	Kunde sendet Assi mit Auftrag im Anhang	Assi 1 verschickt an Assi 2 Mail mit Auftrag	Boss schickt Assi den Auftrag	Spieler in Rolle des Kunden sendet Assi Mail mit Auftrag	Boss ist jetzt Kunde und verschickt Auftrag an Assi	Kunde schreibt Mail+Auftrag an Assi	Person schickt Mail (samt Auftrag) an Meier	Kunde schickt Assi Mail mit Auftrag im Anhang	Meier ist Schmidt geworden, Schmidt schickt Auftrag an Meier
Confidence	–	6	6	7	–	5	–	6	6	7	3	7	7	7	5
Correctness	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*	*Correct*

Table B.2.: German data for the experiment described in Section 8.2.2 (summarized from the free text responses by the students who rated the correctness)