MDE Settings in SAP: A Descriptive Field Study

Regina Hebig, Holger Giese

Technische Berichte Nr. 58

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 58

Regina Hebig | Holger Giese

MDE Settings in SAP

A Descriptive Field Study

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de/ abrufbar.

Universitätsverlag Potsdam 2012
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2012/6019/
URN urn:nbn:de:kobv:517-opus-60193
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-60193

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-192-9

MDE Settings in SAP: A Descriptive Field Study

Regina Hebig and Holger Giese

System Analysis and Modeling Group
Hasso-Plattner-Institut at University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam
forename.surname@hpi.uni-potsdam.de

Abstract

MDE techniques are more and more used in praxis. However, there is still a lack of detailed reports about how different MDE techniques are integrated into the development and combined with each other. To learn more about such MDE settings, we performed a descriptive and exploratory field study with SAP, which is a worldwide operating company with around 50.000 employees and builds enterprise software applications. This technical report describes insights we got during this study. For example, we identified that MDE settings are subject to evolution. Finally, this report outlines directions for future research to provide practical advises for the application of MDE settings.

1 Introduction

Today, software is build in various ways, starting with the simplest case, where a single programming language is used and after a compilation step the developed software can be applied. In more complex cases the developer has to deal with different representations of a system. For example, in the model-driven engineering (MDE) approach a system is first given in an abstract representation, which is refined with more and more information during each development step. Thereby, the language in which the system is represented might change several times. Thus, not a single programming language, but a whole set of (modeling-) languages, tools and their interaction have to be considered than talking about productivity in software development.

Within SAP the combination of languages, tools, and development activities is called *programming model*. However, this name can be misleading, since it is often used as synonym for programming paradigm. We will use in this report the following terms: We use the term *MDE setting* to describe the used combination of tools, languages, automated activities (transformations, generations, validations, interpretation, compilations), and manual activities (modeling, programming, or configuring). We could observe that, automated and manual activities are usually performed in a certain order (customary praxis), which we call *fine granular process*. MDE settings together with the associated fine granular processes are *object of study* of this survey.

Currently, little is known about how MDE settings look like in practice, and how they influence the fine granular process, performance of the development or even changeability of software. However, it is known that MDE settings can also have negative effects [16]. Thus, it is difficult to ensure in practice that an optimal MDE setting is used.

The goal of this exploratory field study is to learn how MDE settings and the corresponding fine granular processes look like in industry on the example of SAP. In short term, documenting MDE settings explicitly, can facilitate discussions and exchanging of experiences between developers. In future the documentation of MDE settings will help us to better understand how the software quality attributes are influenced by MDE settings and to develop support for planning MDE settings as well as changes to them. A further long term goal is to identify best practices and to support the identification of potential for the improvement concrete MDE settings.

1.1 Foundation: influences on productivity

A main motivation in research on software development techniques, technologies, and languages is the desire to enhance the productivity of a software company. As collected in [2, 3] Balzert's Lehrbuch der Softwaretechnik much research was done to understand and define the term productivity, exploring its relation to attributes, such as quality and quantity of software, development effort, and development time.

In [26] Sneed presents his devil's square (Figure 1) to illustrate that it is hard to enhance productivity in the short term. Thereby, the devil's square illustrates productivity as a plane, spanned by the parameters quality, quantity, costs, and time. Sneed assumes that the expanse of the plane, does not change easily. E.g. actions applied to enhance quantity of the produced software within the same time and equal costs will reduce quality of the product. Enhancing quantity of the software without decreasing quality can, in short term, only be reached by adding costs and/or time.

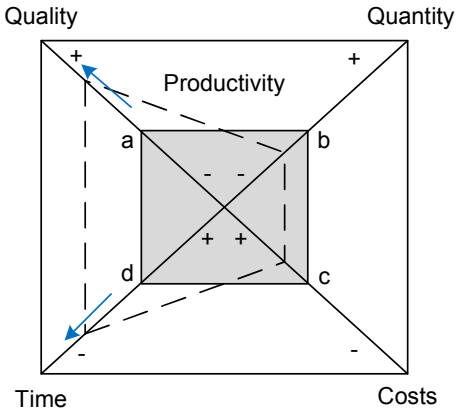

Figure 1: Sneed's devil's square after [26]

Consequently, further work was done, to answer the question how productivity can be enhanced, i.e. how one of the four parameters can be optimized, while keeping the others stable. As cited in [3], Maxwell et al. sums up that intensive usage of tools and modern methods can increase productivity [20]. Further, research was done on the question whether reuse can reduce required time and resources without reducing quality and quantity of produced software [11]. Boehm presents several actions to enhance productivity in [4]. Finally, Balzert subsumes that process improvements might improve productivity [2].

As indicated with Sneed's devil's square, productivity is influenced by other factors, which are presented in [3] on the basis of Basili (see Figure 2). Thereby, productivity is influenced by value of the produced software (i.e. quality and quantity or the software) and costs of the software

(which is among other things influenced by development time and staff costs). Further, complexity of the software to be build influences the level of difficulty, which further influences the costs.

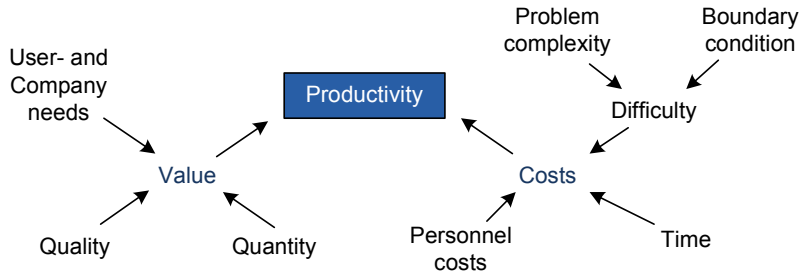

Figure 2: Influencing factors on productivity after Basili (after Balzert [3]).

The introduction of MDE is associated with many well founded hopes concerning improvement of quality, quantity, costs, or time and, thus, improvement of productivity. There are different arguments why MDE or MDA should break through the devil's square, by improving one parameter without worsen the others.

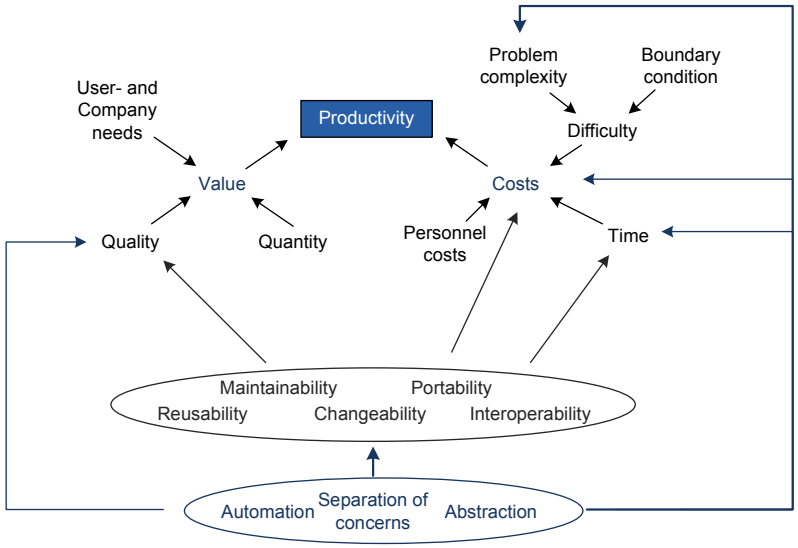

Figure 3: Influencing factors after Basili (after Balzert [3]), extended with software quality parameters and MDE concepts: separation of concerns, automation and abstraction.

Therefore, the model-driven engineering comes up with three concepts, which are *separation of concerns*, *automation*, and *abstraction*. For example, in the OMG's MDA standard [24] the concerns separated are business aspects, computational aspects and platform aspects. Separation of concerns is thereby reached by using different models/artifacts for expressing the different concerns. Abstraction is often associated with easing the definition of certain aspects. These three concepts influence so called software quality parameters, e.g. *reusability*, *maintainability*, *changeability*, *portability*, and *interoperability*, which further influence Basili's influencing factors (quality, quantity, cost, time, and complexity). Figure 3 summarizes the relation between Basili's influencing factors and the discussed MDE concepts.

As McIlroy's law tells us: 'Software reuse reduces cycle time and increases productivity and quality' [9]. Thus, reuse mainly helps to reduce development time, without reducing quality, quantity,

or enhancing costs. Further, reuse can help to improve quality, since faults is reused model or code parts are more probable to get discovered with each reuse. Consequently, reused model and code parts are more mature.

Due to the need that software can be used over a long time, main aspects of productivity are maintainability and changeability. Thus, reducing time and costs for changes and maintenance is important for productivity. Further, maintenance helps to preserve quality of a software product. In addition, *customizability*, a special form of changeability, helps to enhance the validity of the software for different customers. Portability enhances productivity, if the same system has to be build on different platforms, since costs and time can be saved. Automatically achieving interoperability can save time and costs and, thus, can improve productivity.

1.2 Structure of this report

The rest of this report is structured as following. In the next Section 2 we examine related work concerning MDE's influence on productivity. In Section 3 we introduce the design of this study. The results of the study are summarized in Section 4. Next in Section 5 we discuss the results of the study with respect to related work. In Section 6 we introduce general insights that we gained during our study. Finally, we discuss treats to the validity of the studies results in Section 7 and conclude in Section 8, where we further show up resulting research directions.

2 Related Work

There are two kinds of literature dealing with the question whether and how MDE influences productivity. First, we want to have a look at literature that comes up with arguments for the influence of the MDE concepts on software quality attributes, without providing an empirical proof. Afterwards, we have a look at studies and experience reports that aim to prove positive influences of MDE on productivity empirically.

2.1 Theoretical explanations of MDE concept's influence on software quality goals

In literature theoretical considerations to argue why the MDE concepts separation of concerns, automation, and abstraction influence software quality parameters can be found.

Stahl et al. [27] argue that *reusability* can be supported by a meaningful separation of concerns, as a model that is separated from certain aspects can be reused if these aspects change. For example, a main motivation of OMG's MDA [24] is the reuse of the platform independent model (PIM) for building the software for multiple different platforms, since platform aspects are not specified in the PIM. They further argue that automation leads to reuse, since implementation knowledge is reused in automated transformations or generations. Finally, Gruhn [12] argues that raising the level of abstraction on a domain specific layer leads, in addition to reuse of technical knowledge, to reuse of domain knowledge.

Also for *changeability* and *maintainability* Stahl et al. [27] argues for the supporting role of separation of concerns. Thereby, separation of concerns ensures that changes do not have to be applied in all models of the system. As Kelly et al. [17] argues, a setting where platform aspects

are separated and introduced by a fully automated transformation allows reaction to changes in platform requirements, since only the transformation has to be changed and reapplied to all build systems. Further, Kleppe et al. [18] and Gruhn et al. [12] argue that abstract models that are automatically transformed to code, solve the problem of keeping a consistent high level documentation. A good documentation is necessary to ensure maintainability and changeability.

Portability is one of the main declared goals of OMG's MDA [24]. Kleppe et al. [18] argue that separation of concerns and automation enable changes of the underlying platform.

The third declared goal of OMG's MDA [24] is *interoperability*. Kleppe et al. [18] argue that this is addressed in MDA since platform aspects are automatically generated by transformations, and, thus, automated generation of platform bridges, such as 'PSM bridges' or 'Code bridges', is possible on the same information.

Further, Kleppe et al. [18] and Stahl et al. [27] argue that abstraction directly helps to handle *complexity* and they argue that automation has a direct influence on the time required, as each generated part of code, does not need to be written by hand [18, 27]. Finally, as Kelley et al. [17] argue, quality is supported, since DSLs reduce through abstraction the possible mistakes that can be made during implementation.

2.2 Related field studies

There are several studies, dealing with the question how model-driven engineering or model-driven development influences software development. The results of most of them are summarized in the review paper 'Where is the proof?' of Mohaghegi et al. [22]. This paper shows that there is no final proof for most of the common assumptions about MDE's influence. E.g. there are studies supporting the assumption that MDE enhances productivity, while other studies state that MDE reduces productivity. Also a positive influence on software quality is not empirically proven. Several studies were published after this review paper. E.g. in [13] the influence of model size and model complexity on effort during development and software quality is examined on the basis of a case study. However the authors do not focus on the combination of different models or MDE techniques.

Hutchinson et al. performed several interviews to get a broad view of experiences with MDE in different domains. In [15] they present excerpts of three of these interviews. However, they focus mainly on social and organizational factors of MDE. Further this group runs currently a questionnaire [29] to examine how MDE is used today and to identify factors which influence the success of MDE. In contrast to our study, this questionnaire, too, does not focus on the process of applying MDE techniques during development, but of social and organizational aspects.

In [14] the impact of model-driven development on the process is examined on a case study. The authors identified fourteen factors (mainly social and organizational) that influence the process.

Further, in [21] the usage of modeling techniques, methods and languages is examined on a case study in automotive domain. In [28] experiences with MDE in Motorola (domain of mobile devices) are described. They report an improvement of quality and productivity, but also a lack of modeling skills. A further experience paper is [23], which subsumes experiences made in different case studies, one of them also within SAP.

In contrast to the study presented in this paper, most existing studies or experience reports do not focus on the design of MDE settings (i.e. on the interplay of different MDE technologies). Instead, often social and organizational factors are in focus.

3 Design of the Study

This survey was performed in form of a descriptive field study. Our focus is the usage of MDE techniques during development. As indicated in Figure 4 a setting of such MDE techniques (MDE setting) bases on different languages and tools, implementing different transformations, code generation or supporting the developers. Tools and transformations are provided by tool developers and used by developers to create software, which is further used by users.

We are interested in the fine granular processes of applying manual and automated activities, such as model transformation or code generation, during development. In SAP these processes include many activities and are interwoven with each other. Thus, it was necessary to chose perspectives to capture manageable parts of these processes. Therefore, we chose one of two perspectives, respectively. The first perspective is the one of the creation of an important intermediate product (e.g. 'creating a business object'), i.e. a part of the software. The second perspective is the one of a development tool within the MDE setting, supporting specific parts of an implementation process. An example for the latter case is the perspective of the tool Service Implementation Workbench (SIW), which supports developers in generating a web service on the basis of a WSDL file and existing components.

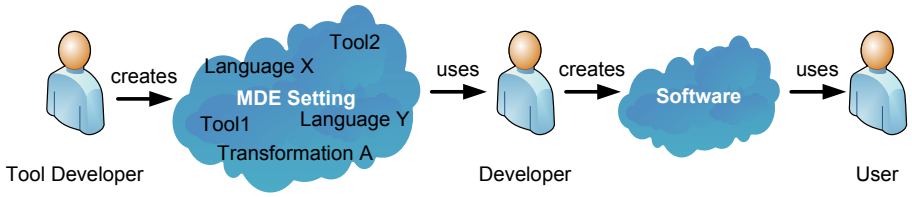

Figure 4: Roles using and creating an MDE setting for the creation of software

As a first step, we had to identify appropriate MDE settings for the study. Therefore, our two contact persons within SAP provided us with a list of different objects. Thereby, they chose MDE settings which are known for including the usage of models and MDE techniques.

As a next step, our contact persons introduced us for each MDE setting to one or two interview partners. Thereby, they either chose persons with experience in development using this object of study (i.e. this part of the fine granular process). In this case, the interview partners are SAP developers or consultants preparing SAP software for the customer. Otherwise, for some objects from the development tool perspective the interview partners are tool developers (or software architects) that built the corresponding tool.

From the objects chosen by our contact persons, we included each one in the study, where we could find one or two interview partners that agreed to support us in the survey. Altogether, we got a set of six objects for this study.

The investigation of each object of study started with an initial telephone interview with the main interview partner (as illustrated in Figure 5). All telephone interviews lasted between thirty and sixty minutes. For the initial telephone interviews we used the technique of structured interviews [1], which was identified to be one of the most effective elicitation techniques in [8]. To provide a frame, we started the interviews with a short introduction of our goals and interests. As recommended for structured interviews we prepared a set of question groups.These question groups cover the following key topics (considering artifacts as models or source code):

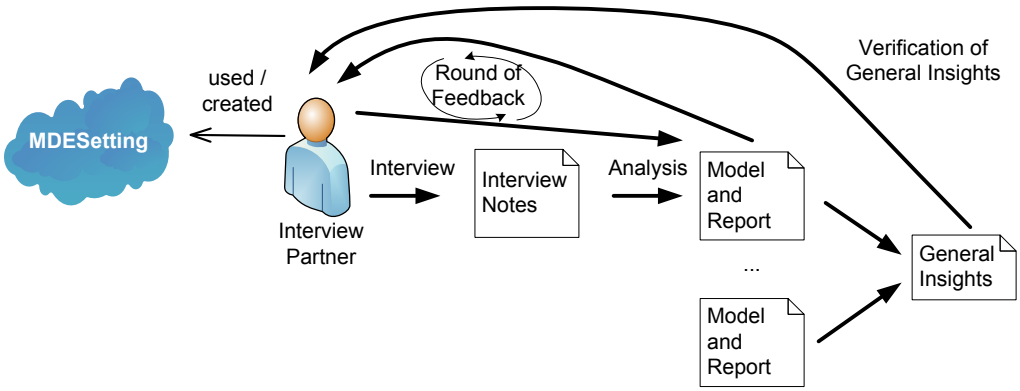

Figure 5: Design of the study

- Used tools as well as modeling- and programming languages
- Artifacts that are used and created during development
- Existing and created relations between artifacts
- Used activities to change, enrich, translate, generate, merge, compile, or interpret artifacts
- Degree of automation of activities
- Order of activities
- Performed (semi-) automated quality assurance activities on artifacts
- Responsible roles for different activities

For four of the objects of study the interview partners provided us, additionally to the interviews, with documentation material, such as usage scenarios. In a next step, we analyzed the information collected in the initial interview and the documentation material. Thereby, we structured the collected information and used UML activity diagrams ([25]) to model a process of the captured activities. We annotated the activities with required input artifacts and produced output artifacts. In addition, we created a textual description of the results, to capture information about how relations between artifacts change, degree of automation of the activities, responsible roles, and which tools and languages are used. Analyzing, structuring, capturing, and modeling the information took one of us approximately between one and one-and-a-half day of work.

Due to several reasons captured information always contains ambiguities and insufficiencies to a certain degree. One example reason is that e.g. the role developer in Figure 4 is not for each object of study an SAP member. E.g. in some cases the tool in the MDE setting is sold as a product by SAP. Consequently, terms like 'developer' or 'product' can be misleading. To handle such ambiguities and insufficiencies, we provided our findings to the interview partners to get feedback. Thereby, we performed between four and ten rounds of feedback per object of study. Each round of feedback required us to formulate appropriate questions to tackle the ambiguities. Subsuming each round of feedback took one of us approximately one day of work, for formulation of questions and adoption of the feedback into our findings.

To evaluate our results, we formulated for each project a report that was provided to the interview partners and discussed our findings in a further telephone interview. In three cases this led to some further small changes in the models. Preparing the final report together with the interview

took one of us approximately another day of work. Subsuming, we spend in average nine days of work on the investigation of one object of study.

Inspired by grounded theory [7], we compared the transcripts and models of the different objects of study. Thereby we identified some more general insights. Finally, we confronted our interview partners with these general insights to validate them. In addition, we used a small questionnaire to ask for motivations of the introduction of the different objects of study and for the software quality parameters that are aspired to be supported.

4 Results of the Study

As mentioned above, we captured six objects of study. During examination of the objects of study, we also captured the fine granular process associated with the MDE setting. As usual for processes, the actual executions vary concerning the number of artifacts to be created, the number of cycles of repetitions or iterations during development and the actual execution order of parallel activities. Furthermore, for two of the MDE settings and their fine granular processes (*SIW* and *BRF*) we captured variations. We found three reasons for that. First, the activities of MDE settings and the fine granular process can vary with the system where the product should be integrated into. Second, the activities and their order can vary with the role performing them (costumer or developer). Finally, the activities and their order can vary with different available artifacts (e.g. does a web service description already exist?).

4.1 Objects of study

The six captured objects of study are summarized in Table 1. First, there is the MDE setting used to develop business objects (an SAP specific type of components that captures functionality specific to a certain business need) for the feature package 2.0. This object of study (referred to as *BO* in the following) describes an old development approach, which was used from 2004 on ca. 100 times in nine teams. Today *BO* is substituted by a new development approach, through substitution of tools. Motivation for the initial introduction of *BO* was to enhance quality, but also to reach transparency and a unified procedure for business object development.

The second object of study is the MDE setting for the development of web services with the tool service implementation workbench (referred to as *SIW* in the following). Since ca. 2003, the tool allows development of a service on the basis of a web service description (formulated in web service description language (WSDL) [6]) and supports reuse of existing functionality. Goal for the introduction was the prevention of redundant implementations and the corresponding effort for maintenance. *SIW* is used throughout the whole SAP Business Suite. By today *SIW* was partially redesigned.

Further, objects of study describe the development of user interfaces and there coupling to services or business objects. On the one hand, there is the development of a user interface for SAP Net Weaver applications using the tool visual composer [5]. This object of study is referred to as *VC* in the following. *VC* is used since 2006 ca. four times at the consulting company FIVE1 GmbH & Co. KG. Motivation for its introduction was a faster development as well as the ability to create prototypes. Between the different versions *VC* was often extended.

On the other hand, there is the development of a user interface and application based on business objects using the tool Oberon. This object of study is referred to as *Oberon* in the following. *Oberon* was used since 2008 ca. 120 times. Motivation for the introduction of *Oberon* was the

simplification of user interface technologies that are used in context of By-Design. In addition, the flexibility was enhanced through usage of similar models at design time and runtime. In this context, the number of tools was reduced and different areas of development were integrated into one tools.

The fifth object of study deals with configuration of business processes using the tool business rule framework (BRF+). This object of study is referred to as *BRF* in the following. The customer specific configuration can be seen as part of development of software systems, too. However, *BRF* can be applied by experts at the customer's side. Since 2008, *BRF* was used at least 70 times within SAP standard applications and adapted multiple times. Motivation for the introduction was cost reduction, reaching transparency and agility of development.

The last object of study allows the definition of a mapping between data from different tools. The extraction of data, mapping, and generation of a HTML report is then performed automatically and periodically during runtime. The tool that supports this definition and automated execution is Business Warehouse. Therefore, this object of study is referred to as *BW* in the following. Since 1996, *BW* has ca. 19 000 customers. *BW* was introduced to substitute the predecessor EIS (Execution Information System), with the motivation to enhance performance and to enable consolidation of data from multiple systems. By today, *BW* was changed multiple times, according to changes of the underlying language (ABAP) or to integrate different transformations.

Object of Study	Full Name	Approximate number of applications
BO	Development of Business Objects for the feature package 2.0	ca. 100 times in nine teams
SIW	Development of web services using the tool service implementation workbench	used throughout whole SAP Business Suite
VC	Development of a user interface for SAP Net Weaver applications using the tool visual composer	ca. 4 times at FIVE1 GmbH & Co. KG
Oberon	Development of a user interface and application based on business objects using the tool Oberon	ca. 120 times
BRF	Personalization of business processes using the tool business rule framework	at least 70 times within SAP standard applications
BW	Definition and automated execution of reporting with the tool BW	ca. 19 000 customers

Table 1: Objects of this study

4.2 Artifacts

In the observed MDE settings different types of models and artifacts are used. However, it is difficult to count the number of used languages for two reasons. First, it is not in each case easy to determine whether something can be count as language. The main reason is that not each language has an explicit meta model, such as the rule set definition language of *BRF*. One of our interview partners get to the heart of it by asking whether something that is only supported by a single tool is a language. Another example is a setting were specific artifacts are called 'models' explicitly, but interview partner would not call the formal definition of the artifacts structure a language definition. In consequence, some languages have no explicit name (e.g. the user interface models in Oberon have no official name).

Second, it is not easy to determine what category a language is of (i.e. general purpose language (GPL), domain specific language (DSL), textual or graphical modeling language). For example, one of our interview partners rated Excel as GPL, while another did not. One answer just explicitly

excluded DSL. One language was described as a textual modeling language which is graphically modeled. WSDL (web service description language) was not rated as belonging to one of the above mentioned categories ('WSDL defines messages, not their content (objects)' (transcription translated from German)). ABAP was rated as programming language or GPL. In one case the answer was 'neither of them [GPL, DSL, or modeling language], programming language; most likely GPL' (transcription translated from German) This illustrates that the usually used terms are not sufficient to express differences that developers experience. Finally, HTML was barely rated as DSL ('can be called a DSL (enforcedly)' (transcription translated from German)) by the interview partner, who communicated that he rather would call HTML a rendering technology.

The following results base on the rating of our interview partners (shown in Tabel 2). We found two general purpose languages or programming languages: ABAP and Excel (which in one case rated as GPL and in one case not). Further, we found five domain specific languages (DSLs): HTML, status and action models (S&A Models), UI Component Definition Language, MDRS language for meta data, and Business Object Definition Language (BODL). We found five modeling languages: BPMN, Generic Modeling Language (GML), data-flow diagrams, UML Plus, and ESR specific Business Object Model. Finally, we identified four further artifact types: WSDL artifacts (web service description language), context variables with DDIC-Reference (Data-Dictionary), Info Objects (defined in ABAP Dictionary), and rulesets (Regelsprache und Modellierungssprache).

Object of Study	GPL / Programming language	DSL	Modeling language	No category assigned
BO	• ABAP • Excel files	• S&A	• UML Plus • ESR specific meta model for business objects models	
SIW	• ABAP			• Context variables with DDIC-Reference • WSDL
VC			• GML • BPMN	• Excel files
Oberon		• UI Component Definition Language • MDRS • BODL		
BRF	• ABAP		• BPMN	• Regelsprache und Modellierungssprache
BW	• ABAP	• HTML	• data-flow diagrams	• Info Objects (defined in ABAP Dictionary)

Table 2: Used languages

4.3 Activities

We captured between ten and twenty activities per object of study. During this elicitation we captured activities of different degrees of detail. Thereby, we differentiate between minimal changes or transitions and complex changes or transitions. Within a minimal change or transition only one or a hand full of connected model elements are changed or transcribed to another artifact, respectively. In contrast, within a complex change or transition a potentially big part of the model is changed or transcribed to another artifact, respectively. Activities that represent minimal changes and transitions are often described in manuals and tutorials, e.g. for learning to handle a tool. However, for getting an overview and analyze an MDE setting they might be too fine granular. An examples for such minimal activities in the study is the creation of a floorplan using Oberon.

Here, a standard floorplan is automatically initialized in form of a template that can later be filled with information. During the study, we captured ca. 28 of such minimal activities.

Considering the more complex activities we captured nine fully automatically supported generation and transformation activities. This contains automated activities, where a complex change is performed on a model (or artifact), or automated activities, where on the basis of one or more input models (artifacts) at least one output model (artifact) is created or manipulated with a complex transition from the input models to the created/manipulated model. Thereby, we count variants of an automated transformation or generation as one activity in this statistic. Further, we captured eight complex semiautomated generation or transformation activities, e.g. the generation of a proxy using the Service Implementation Workbench (SIW), and one complex semiautomated template instantiation.

In addition, three of the objects of study contain, besides tests, three fully automated and three semiautomated verification activities. In *SIW* it is verified that a mapping between service descriptions in different languages is complete. In *BO* a set of checks is applied on the business object model. Finally, in *BRF* certain analysis steps are applied of the rule sets. In addition, the *BRF* rule sets can be simulated.

Further, we wanted to know, whether the resulting system implementations are compiled or interpreted (results shown in Table 3). Interestingly, the answer was only in two cases clearly 'compiled' (*SIW*) or 'interpreted' (*Oberon*). Within the object of study *BO* the ABAP code is compiled, but the additionally used S&A models are interpreted.

For the user interface in *VC* it depends on the use case, whether interpretation is sufficient or compilation has to be performed. For *BW* it was reported that a mix of both, compilation and interpretation, is used. Thereby, the share of interpretation grows over time, as the dynamic parts of the underlying ABAP increased. Similarly, for *BRF* we got the answer that a mix is used for the created ABAP code as well as for the created rules.

Object of Study	# automated generation and transformation activities	compilation / interpretation
BO	1	ABAP code is compiled ; S&A models are interpreted
SIW	3	compiled
VC	2	mixed (depends on the use case)
Oberon	0	interpreted
BRF	2	mixed
BW	1	mixed

Table 3: Distribution of automated generation and transformation activities as well as distribution of compilation and interpretation

4.4 Occurrence of MDE concepts

As described in Section 2.1 MDE comes up with three concepts, which are separation of concerns, automation, and abstraction. We searched for occurrences of separation of concerns, automation, and abstraction in the captured objects of study. As there is no measurement how more abstract a language is compared to another language, we could not captured to which degree abstraction is used. Further we have no measures to which share of the resulting artifacts is generated. Thus, we used very simple criteria to locate the three MDE concepts, based on the capture artifacts and activities.

Therefore, we first need the notion of an *automation boundary*, which denotes an automated activity or set of automated activities that are not followed by any manual activity to retrieve an executable product and consume artifacts that are not created fully automatically. In classical code centric development, this automation boundary is e.g. the compilation. However, there are only some MDE approaches where no automated activities exists 'before' the automation boundary.

It becomes clear that all three MDE concepts already influence each other. Intuitively abstraction might be understood as the difference between artifacts before and after the automation boundary, as information that occurs only in artifacts behind this automation boundary is abstracted away from the developers perspective. However, generation or transformation steps might occur in MDE already before the automation boundary (i.e. followed by manual activities that work on the basis of the automatically produced output). This makes the occurrence of abstraction more diffuse. Thus, we further considered where content is moved (automatically) between two artifacts and whether the different roles of developers involved in the MDE setting. We consider separation of concerns as parallel existence of artifacts that might reference each other, but are not created on the basis of each other.

4.4.1 Separation of concerns

All object of study contain examples for separation of concerns. In *BO* behavioral parts of the business object can be defined separately in S&A models, while the rest of the system in implemented in ABAP code. The separation is used to enable that certain parts of the business object can be implemented at a higher layer of abstraction (i.e. the parts that can be implemented using S&A models are separated from the parts that have to be implemented in ABAP).

In both, *VC* and *Oberon* the user interface models are separated from the functionality using the service concept. Additionally, in *Oberon* the basic UI model is separated from change transactions, that define changes on the UI model. In the *SIW* the development of a service is separated from holding the corresponding web service description consistent in the enterprise service repository (ESR). Thus, service description is separated from the code. *BRF* separates the rule definition from the process definition. Finally, in *BW* the info source is separated from the transformation description for the data, which is further separated from the layout definition. Here the separated artifacts reference each other.

4.4.2 Abstraction

In *BO* several forms of abstraction are used. Initially, ARIS models are used to specify parts of the system. On these models also some verification activities are performed. Later these ARIS models are transformed to ESR models, which are automatically generated to the less abstract ABAP code. Further, S&A models are used to describe parts of behavior more abstract than with ABAP code. In *VC* a BPMN model can be used to generate an initial version of a user interface model, which can be counted as abstraction. In both, *VC* and *Oberon*, the user interface models are quite abstract representations. However, it is difficult to measure on the basis of the given data how abstract they are.

In the *SIW* a proxy is generated on the basis of the more abstract WDSL file. Further, in *BRF* from the perspective of the user the process definition is abstracted away, as they have only to deal with the rule definition language. However, this abstraction is to be implemented manually by the developers that define where in the process the business rule service is called. Finally, in

BW the InfoSource might be seen as an abstract view of the technology specific data abstraction. As in *BRF*, however, this abstract representation is implemented manually.

4.4.3 Automation

In *BW* and *BRF* we captured automated activities. However, these automated activities are located at or behind the automation boundary. Similarly, in *Oberon* we captured no automated activity before the automation boundary. The user interface models there are directly interpreted. In *VC* the generation of a UI model on the basis of a BPMN model is located before the automation boundary. The generation of code as well as the generation of a WSDL file in the *SIW* are both located before the automation boundary. Finally, in *BO* the generation of code and the instantiation of templates are located before the automation boundary.

4.4.4 Length of activity chains

Finally, we have a short view on the lengths of the activity chains, i.e. the number of activities that have to be executed when an artifact is changed in order to retrieve an executable system again. Thereby we count only activities before the automation boundary. In *BO* there are quite many activities that have to be performed when the ARIS model is changed (up to eight activities plus three to four activities for local test and integration test). Also in the *SIW* the number of activities that have to be performed again if e.g. a WSDL file changes, is quite high (five to six activities). In contrast, in *BW* changes in the most artifacts do not require the execution of many activities (including the application of the change: one to two activities). Only changes in the container artifact require more complex changes (up to five activities). The number of activities that have to be executed if a process is changed in *BRF* is quite long, too (four to six activities, performed by different roles). However, again for specific artifacts - in this case for rules and decision tables - the number of activities that have to be executed again is low (one activity for applying the change and one to three checking activities). For both, *Oberon* and *VC*, changes can be directly introduced into the system (except for some complex cases), as the user interface models are directly interpreted or compiled, respectively.

4.5 Software quality parameters

As a longterm goal of this study is also to better understand the influence of MDE settings on software quality parameters, such as changeability or portability, we further asked our interview partners whether different software quality parameters hold for the different settings.

Changeability or the ability to easily change certain parts of the system was indicated for all six objects of study. Thereby, for *BO* the ease of change was evaluated by our interview partner as restricted to elements, since many changes lead to additional implementation effort. For *BRF* agility was named as motivation, while the corresponding interview partner named decision tables as artifacts that can be easily changed. For *BW* the high number of APIs (application programming interface), was named as a reason for a high changeability.

Maintainability was indicated for four objects of study, which are *VC* (since version 7.1+), *SIW*, *BO*, and *Oberon*. Reusability or the ability to easily reuse certain artifacts or parts of the system was indicated for all six objects of study, too. Thereby, for *VC* reusablility was named for version 7.1+ or higher. For *BW* the DataExatractors and InfoSources are named as easily reusable artifacts. For *BRF* the reusability is indicated for decision tables.

For four of the objects of study customization or personalization, respectively, were indicated. Thereby personalization is similar to customizability, but special to single persons instead on whole customers. These objects of study are *Oberon*, *BW*, and *BRF*. For *VC* personalization was named as goal since version 7.1+. For *BRF* personalization was rated as an underlying aspect.

Interoperability was named property for *VC* and *SIW*. Finally, portability was indicated for *BO* as well as for *Oberon*, where the models were already interpreted for different platforms (e.g. Blackberry, iPhone, Android, HTML, and Silverlight). The described data are summarized in Table 4. It has to be noted that this list does not indicated or include information to which degree positive effects on the different software quality parameters are experienced in practice.

Object of Study:	BO	SIW	VC	Oberon	BRF	BW
Changeability	✓	✓	✓	✓	✓	✓
Maintainability	✓	✓	✓(since version 7.1+)	✓		
Reuse	✓	✓	✓(since version 7.1+)	✓	✓	✓
Customization				✓	✓	✓
Personalization		-	✓(since version 7.1+)	✓	(✓)	
Interoperability		✓	✓			
Portability	✓		-	✓		

Table 4: Software quality parameters indicated for the different objects of study. The minus sign indicates that our interview partners explicitly denied that these properties hold.

4.6 Subsumption

Subsuming, there is a mixture of languages and models used during development and MDE techniques such as generation, transformations, or verification are applied. There are objects of study where models are compiled to code, while other objects of study contain a direct interpretation of the models or even a mix of both. Nevertheless, the automated activities are in some objects of study interwoven strongly with manual activities (i.e. automated activities occur before the automation boundary). For example, while the business object model (formulated in UML Plus) is automatically tested in *BO*, it has to be translated manually to the ESR specific meta model afterwards. Other objects of study contain a better separation of activities. E.g. in *BW* the manual activities for defining the mappings are applied first and the automated activities (extraction of data and generation of HTML) are applied afterwards during runtime. In Appendix A the detailed descriptions for single objects of study can be found.

5 Discussion

In the following we discuss the above presented results with respect to related work. Thereby, we first discuss whether theoretical explanations for MDE's influence on productivity are sufficient to explain our results. Next, we compare our results to some of the findings of related studies.

5.1 Discussion of correlation between occurrence of MDE concepts and aspired software quality goals

Here we want to discuss, whether the influences between MDE concepts (separation of concerns, abstraction and automation) and the software quality parameters that are described in related work (Section 2.1) are sufficient to describe how the captured objects of study support the indicated software quality parameters.

For all objects of study changeability was indicated as an software quality parameter that is positively influenced. Thereby, in four of the objects of study for the parts of the system that should be changeable two observations hold. On the one hand, these parts are separated into extra artifacts. On the other hand the length of the activity chain is very short (i.e. not much activities have to be reapplied when the change is applied). This holds for *BW*, where the layout might be changed easily independent of the data extraction, *BRF*, where rules and decision tables can be changes without accessing the rest of the process, as well as for *VC* and *Oberon*, where the models of the user interfaces are separated from the functionality and can be changed directly. Interestingly, within *SIW* the number of activities that have to be applied if e.g. a WSDL file changes is quite high (the proxy has to be regenerated, than the code has to be regenerated and further adapted manually). Nonetheless, changeability is claimed as property of the *SIW*. Looking further into this discrepancy we learned that the generation steps within the *SIW* are implemented such that they support protected regions. Thus, in case of the application of a change, not the whole manual adaption of the generated code has to be implemented newly. Thus, it seems that *SIW* includes mechanisms to reduce potentially negative influences on changeability. In contrast, although changeability was named as property for *BO*, it was admitted that this only holds for some element, while other changes require further implementation effort. This fits with the observation that the length of activity chains to apply a change e.g. to the BO model in ARIS within *BO* is quite long.

Maintainability was indicated for *Oberon*, *VC*, *SIW*, and *BO*. Thereby, maintainability which is a special case of changeability, seems to manifest in the MDE settings similarly, i.e. mainly by short lengths of activity chains. As mentioned above this holds for *Oberon* and *VC*, while in the *SIW* the protected regions help to reduce the effort than applying a change. Again it seems for us that maintainability is only partially supported for *BO*.

Customizability and personalization, where named as properties of *Oberon*, *BRF*, and *BW*. In all three cases, the parts that can be customized or personalized are specified in separate artifacts which trigger short activity chains. Thus, like maintainability, customizability and personalization can thereby be seen as special cases of changeability. In *BW* the layout can be customized, in *BRF* the rules and decision tables are separated from the process and in *Oberon* change transactions can be defined parallel to the basic user interface model and are consumed by the interpreter. For *VC* from version 7.1+ on personalization was indicated as property. Thereby, changes in the user interface model in *VC* require a low number of activities to be performed.

It can be seen that customizability, personalization, changeability and maintainability cannot really be differentiated from the perspective of MDE settings. The difference rather lies in the questions what parts of the system are implemented in separate artifacts that can be changed without triggering much changes in other artifacts.

Similar to the argumentation of Stahl et al. [27] all four software quality parameters are influenced by the separation of system parts over different artifact. However, in addition the length of the activity chain of changes on different artifacts seems to play an important role. Further, the effective influence depends on the question whether an artifact will probably change.

In addition, as also Heijstek et al. [14] argue the complexity of the tool chain influences also maintenance ('Maintainers need to be educated during development to be able to understand

the modeling and generation process' [14]). That MDE can also lead to problem with changeability is known. There are works that address the problematic of overwritten code by reapplied generations (e.g. generation gap pattern by Fowler [10]).

Similar to changeability the ability to reuse parts of the system was indicated for all objects of study. In two examples, *SIW* and *BO* this can directly be seen in the MDE settings, where we captured explicit steps for the usage of templates. In the other four cases the system parts that are listed to be often reused are well separated in own artifacts. In *BW* Info Sources and DataExtractors are often reused in practice, in *BRF* decision tables are reused, in *Oberon* user interface components are reused and in *VC* reuse of services happens.

Reuse, similar to the different forms of changeability, depends on the question how different concerns of the system specification are separated over different artifacts, which fits to the argumentation in [27].

Interoperability was indicated as property of *SIW* and *VC*. Thereby, in both approaches it seems to the usage of the service concept that enables interoperability. Thus, *VC* allows accessing functionality via a couple of protocols. The *SIW* was initially used to build web service adapters for functionality provided via BAPI interfaces.

Interestingly, interoperability was in both cases not reached by generation of platform bridges (as argued in [18]), but through the usage of the service concept.

Finally, portability was indicated for *Oberon* and *BO*. In *Oberon* this is reached as the user interface models abstract over platform details and can be interpreted by different interpreters. This is conform to the arguments for a positive influence on portability of Kleppe et al. [18]. For *BO* it turned out to be difficult to identify a reason for the portability in the MDE setting. Probable reasons for that might be, that the scope of the parts we captured does not cover how artifacts might be used prepared, such that the system can run on different platforms.

Subsuming, there are multiple other aspects influencing the software quality parameters, e.g. the degree of abstraction, the question what changes are required and probable, or the share of generated code. Thus, the statement of Kelly et al. still holds: 'The exact productivity gain, however, is often difficult to measure' [17]. That it is not well understood what abstraction really is concerning an MDE setting can be seen on the example of automated template instantiations for reusing models. It is hard to rate whether this is a form of abstraction (is the content of the template really abstracted away for the developer) or separation of concerns, where the reusable parts are separated from the non-reusable parts. The same holds for a transformation where the result is further manipulated manually. Can the automatically created parts be rated as abstracted away, if the rest of the artifact has to be touched manually? Another example is the generation of interfaces for existing code. Here the automatic generation is no hint for an abstraction from the perspective of a developer. Rather automation helps here to get artifacts that represent different concerns compatible. Finally, in most cases not many artifacts and activities after the automation boundary are captured in the models. This makes it further difficult to evaluate the degree of abstraction.

However, as mentioned also in [16], the combined application of two MDE techniques, both leading to a better abstraction and reuse of code, can lead to disadvantageous situations. Thus, a future goal is to predict potential negative influence of the MDE setting on changeability and reuse. For example, in *BO* this led to the situation that a manual translation of the models between two languages and tools was necessary. However, such a manual translation is error prone, which counteracts the original intention of supporting quality. Today *BO* is substituted by another MDE setting that reduces this problem. However, *BO* was used in several projects before the improvement was performed. This example illustrates that learning more about these influences is necessary for aiming at optimized MDE settings.

5.2 Discussion of compliance with results from related studies

In [16] Hutchinson et al. present insights into practically applied MDE collected in a large empirical assessment. Among other things they subsume that ‘a lot of MDE success is hidden’, since often automation and modeling is used in a pragmatic way (‘much of MDE in industry is the kind of modeling and/or automation that represents pragmatism in the face of an otherwise tedious or intractable problem.’). During this study we got a similar impression. None of the captured objects of study follows standard approaches such as MDA [24]. Even so, all approaches work with far more abstract views of the system than plain code.

In their review paper [22], Mohaghegi et al. subsume that ’Combining MDE with domain-specific approaches and in-house developed tools has played a key role in successful adoption of the approach in several cases.’ The results of our study support this impression. Most objects or our study, which are successfully used in practice, combine models and domain specific languages and work with in-house developed tools.

In addition, Mohaghegi et al. [22] show up two lacks in the reviewed reports. First, they state that portability to multiple platforms has often not been feasible. Here our study reveals an example that clearly allows portability: the tool *Oberon*, where the models of the user interfaces can be interpreted using different interpreters, which allows porting the application to desktop PCs as well as mobile devices. The second lack identified in [22] is the missing of examples for executable models. Their development is in [22] discussed to be still a challenge. However, the above mentioned tool *Oberon* is also a positive example for executable models. A second example, our study revealed, is the usage of S&A models, which are interpreted, too. Thus, it seems that portability and execution of models is possible for use cases with a very clear focus on a specific domain.

In [23], Mohagheghi et al. state that ’There is no tool chain at the moment and companies must integrate several tools and perform adaptation themselves.’ This is something we experienced for the here captured objects of study, too. In none of the examples an external tool chain was adopted. *BW* might be seen as a tool chain that is provided by SAP and adopted by the customers. However, our interview partner told us, that often only the extraction part of the *BW* setting is adopted.

In [21], a situation is reported, where model are used fragmented without a ’controlled sequence of models and transformations in the process’. In contrast, the objects of our study come along with a sequence of activities (the fine granular process), which is applied similarly in several projects. This different result might be explained with the different domain focused by our study.

Finally, Weigert et al. [28] reports that ’MDE encompasses all phases of the software-development life cycle’. Our study supports this result, as we identified examples where an MDE setting supports the definition of changes for personalization of software after deployment in *Oberon*.

6 General Insights

During comparison of transcripts and models of the different objects of study we gained several more general insights in the usage of MDE techniques within SAP. In this Section we summarize and discuss these insights (Table 5) and provide examples for them.

Number	Hypothesis
1	An MDE setting can strongly constrain the order of activities, e.g. to apply a change to the system.
2	MDE settings actually do evolve permanently
3	Companies that traditionally work with own languages and tools tend to develop company specific MDE settings, instead of using standard tool chains.
4	An MDE setting is reused over multiple projects
5	Single MDE techniques are partly used in multiple phases of the software development life cycle.

Table 5: Insights identified during this study

6.1 An MDE setting can strongly constrain the order of activities, e.g. to apply a change to the system

A first important insight is that the usage of MDE techniques seems to enforce a certain order of fine granular activities during development - independent of the underlying software development process. This happens due to two factors. First, MDE activities, such as model transformations or code generations create not only new artifacts, but, further, change the relations between artifacts. In addition, such MDE activities might even require certain artifact relations for a correct execution. Thus, artifact relations, such as explicit references, but also containment relations, can enforce the execution of one activity creating a relation that is prerequisite for another activity. For example, the MDE setting *Oberon* allows not only the creation of UI designs that are later interpreted. In addition, the creation of so called 'change transactions' is possible. These define changes in the UI design that can be used during interpretation, to change UI for example according to the role of specific user. Thereby, the 'change transactions' has to reference the part of the UI design that has to be changed. Thus, the creation of such a reference is necessary before the interpretation can be performed. Another example can be found in the development of business objects using the tools ARIS and ESR. The business objects are partly implemented by code (generated on the basis of models) and 'status and action models' (S&A models), which are interpreted. In order to enable the execution of the business object it is necessary to create references between code and S&A models. Each of our objects of study contains examples for such required references.

The second factor for an enforced order is the flow of content. In MDE parts of the target system are initially formulated using models. Thereby, the use of abstract modeling languages reduces complexity for the developer. However, often the model cannot be used to express all aspects of the system. Thus, it is necessary to move the modeled content to another artifact. This might happen manually or automatically. The later version naturally should lead to a reduction of errors that can be made by the developer, but also to time saving and reduction of effort. Thus, developing software using different artifacts enforces a flow of content between the artifacts performed by different activities. For example, in *BO* the business object model formulated in the language UML Plus using the tool ARIS has to be translated to an ESR model. While the ARIS model is subject to automated checks, only the ESR model can be generated to code using the tool BOPF. Examples for content flow can be found in each of the six objects of study of this study.

We asked six interview partners whether they agree with the hypothesis 'An MDE setting (a given set of tools, languages, as well as provided transformations, generations and interpretation activities) can strongly constrain the order of activities, e.g. to apply a change to the system.'. Four of the interview partners agreed and the other two answered that they can imagine that this hypothesis is true. Looking at the captured objects of study, all of them seem to support this hypothesis, as they all contain activities that can only be executed in specific orders.

6.2 MDE settings actually do evolve permanently

The second insight we got during the study is that MDE settings do evolve slowly but permanently. This topic caught our attention, as some of our interview partners told us stories about happened evolution steps. Thus, we further tried to find out, why tools were introduced.

Based on the interviews with the developers and the motivations for the creation of the captured tools, we can identify two basic tendencies for changes. In addition, there are plausible reasons, why such changes are helpful and will occur in future, too.

The first tendency is that MDE settings and with them the chain of activities performed during development are step wise changed and enhanced with additional tools, transformations, generations, and manual activities over time. As an example, the Service Implementation Workbench (SIW) was build to generate services for about three hundred ABAP application instead of reimplementing them. Thereby, new MDE activities such as generation of the proxy are introduced to the MDE setting.

Another example is the tool BRF+, which was introduced to eases the adaptation of business processes. Therefore, *BRF* enables the customers to define rules for the execution of their business processes. *BRF* encapsulated the technical integration of the business rules in the process. However, it leads to a longer chain of activities for business process implementation. E.g. for business rule application the business process implementation has to be prepared, such that the rules defined in BRF+ are evaluated at right point in the process. In addition, it is sometimes necessary to define default rule sets.

The captured version of Business Object development (*BO*) is an extension of the predecessor version. Here the tool ARIS was introduced into the MDE setting.

A final example is *BW*, where the tool Business Warehouse was introduced as a substitution of the predecessor tool EIS (execution information system). Reason for this substitution was besides performance the wish to be able to consolidate data from different enterprise systems. Thus, BW is in contrast to EIS not an inherent part of the SAP enterprise system R3 and allows to extract and integrate data from different enterprise systems. Therefore, the MDE setting was extended with activities to define the structure of the data that has to be extracted (info source).

Besides these examples, there are plausible reasons that a growth of an MDE setting happens in future, too. The main reason is that there will be changes in the requirements on the types of systems usually build using an MDE setting. An example for such a change is the upcoming need to build functionality in form of web services, which lead to the creation of the SIW. It is reasonable to assume that there will be the interest to support the implementation of the new requirements on a high level of abstraction, providing automated generation of code parts. However, it may be cheaper and faster to add therefore, a new tool and lightweight language in the MDE setting, than to change the old ones. Further, integration of new development tools can enforce the introduction of transformation activities, to translate the models between different languages. Also the need to ease customization of software can lead to changes in the MDE setting (as with BRF+).

The second tendency for changes in MDE settings is the integration of several tools or automated activities. Since, the first tendency leads to more and more complex MDE settings, it is necessary to also reduce complexity. We identified three examples, where practitioners decided that their MDE setting is too complex and initiated an integration.

The first is the change of the MDE setting *BO*, as it is described in this study. It is no longer in use, since it was substituted by a new version, where the tool MDRS was introduced. MDRS substitutes the tools ARIS and ESR and, thus, shortens the setting and the associated chain

of activities (e.g. the manual translation between ARIS and ESR model is no more necessary). Also the tool *Oberon* was introduced to shorten an MDE setting. First, *Oberon* aims to enhance flexibility by using the same models for design and runtime. Thus, necessary generation steps between the different models could be omitted. Second, *Oberon* was introduced to integrate and harmonize functionality of different user interface technologies that were applied in the environment of By-Design development. Finally, for *BW* it was reported that in some cases different transformations were joint to resolve inconsistencies between semantics of different translations.

A plausible reason why integration will be necessary and happen in future too is, that growth will happen in future too. A complex MDE setting might not only affect internal goals such as performance of the development negatively, but makes the training curve steeper for a new developer. In addition, a long chain of activities is a threat to changeability of a system, since a new execution of a simple generation or transformation step might lead to loss of changes that where applied on the former version of the result.

Both tendencies together illustrate that Lehman's laws ('A system that is used will be changed' and 'An evolving system increases its complexity unless work is done to reduce it.' [19, 9]) seem to apply to MDE settings, too.

We asked our interview partners whether they already experienced these tendencies and whether they can imagine motivations for changing an MDE setting. We got answers from three of the interview partners. All three of them affirmed that they already experienced the change tendencies. Asked for possible motivations to change an MDE setting, they referred to software quality attributes, such as quality, transparency, usability, reduction of development effort, and omit redundant tasks. One of the interview partners pointed out that changes also happen to extend the application scope starting from a prototypical project. This last answer illustrates that MDE settings have to be changed with changing requirements on the produced software.

Finally, we asked our interview partners whether they agree with the statement 'MDE Settings (a given set of tools, languages, as well as provided transformations, generations and interpretation activities) change over time.' We got five answers. Two of them agreed and two can imagine that this hypothesis is true. The fifth interview partner cannot imagine this hypothesis is true for the core of the MDE setting, but agreed for detailed changes.

6.3 Companies that traditionally work with own languages and tools tend to develop company specific MDE settings, instead of using standard tool chains

Further, we got the insight that companies that traditionally work with own languages and tools tend to develop company specific MDE settings, instead of using standard tool chains. An example for that is the MDE setting *BO*, which is specific to ABAP. The user interface development with *VC* and *Oberon* are specific to SAP environments, too. Although the interview partners can imagine conditions for use of both tools outside SAP. Interestingly, the *SIW* was initially build for a use case depending on ABAP. However it was build in a way that other use cases allow their usage outside SAP. In contrast, although *BW* is build in a way that allows its usage with non-SAP enterprise systems, it is practically mainly used with SAP enterprise systems. This is due to the fact that the main value lies in the highly specialized data extraction for the SAP system R3.

One possible reason for company specificness is caused by the above introduced hypothesis that MDE settings actually do evolve permanently. Thus, an MDE setting can result from evolution of a development that already bases on a company specific language. Further, in companies that traditionally work with own languages and tools, there is probably a consciousness for addressing

special needs of the product portfolio with the development environment. However, the reason for company specific MDE settings might be trivially that there is a lack in the provided standard tooling [14]. For example, confronted with the hypothesis of company specific MDE settings, one of our interview partners told us, that they actually evaluated to use a standard meta model instead of their own one. However, they found that this standard meta model was not detailed enough for their needs.

When we asked our six interview partners, whether they agree with the statement that 'Companies that traditionally work with own languages and tools tend to develop company specific MDE settings, instead of using standard tool chains', three of them agreed and one answered that he can imagine that this hypothesis is true. The remaining two answers were given not in common, but only relating to the respective objects of study. One of these cases was mentioned above. There they decided against using a standard meta model. In the other case the interview partner disagreed for the tool that was focus of the corresponding object of study. As explanation, he mentioned that users, for cost reasons, often adopt generation templates for these tool as they stand.

6.4 An MDE setting is reused over multiple projects

Although MDE settings change over time, they are often reused, as illustrated in Section 4. This holds also for company specific settings. For example, the MDE setting *BO* was long time a standard procedure and used in around 100 projects, while the *SIW* was built to build services for approximately three hundred BOR/Bapi functionalities. None of the objects of our study was used only once or even only a hand full of times (note that the usage numbers given for *VC* are only the usages attended by the consulting company FIVE1 GmbH & Co. KG).

6.5 Single MDE techniques are partly used in multiple phases of the software development life cycle

Our observations in the study go a step further than [28], as we found that that several MDE techniques are used in multiple phases of the software development life cycle. For example, models of the user interface design in *Oberon* are partly already used during design phase, before the actual implementation starts. Another example is *BW*, where the interpretation of the reporting (i.e. the generation of HTML) is performed during runtime. Also change transactions in *Oberon* can be defined and added to the system at runtime.

We asked six interview partners, whether they agree with the statement 'Several activities (such as changing a model or generating an artifact) are used in multiple phases of the software development life cycle.' Fife of the interview partners agreed. The other one disagreed, whereas this answer was not given generally, but with the specific object of study in mind.

7 Threats to Validity

Whether the insights we gained in this study are valid for all MDE settings is still an open question. Although six objects of study are fine to get qualitative insights, they are not enough to build an empirical basis. In addition, another threat to the overall validity of our insights is that we only

examined SAP projects in this study. Thus, we cannot make statements about MDE settings in other domains of software or for differently sized companies.

There are some further threats to validity. First, the choice of the objects of study was driven by our contact persons. Thus, the captured objects of study are more probably considered as successful and good example in the company. In addition, many of the objects of study are related to service oriented systems, which can influence e.g. changeability and portability, too. This has to be considered, when talking about influences of the MDE settings on software quality parameters.

A further threat is that five of the six objects of study are capture from the perspective of one development tool, respectively. Thus, the diversity of tools and languages that have to be used to develop a product is in average probably higher than described in Section 4. This high amount of tool perspectives also might have influenced the interview partner's feedback on our hypothesizes. For example, some of the answers were given not for a common situation, but with specific tools in mind.

8 Conclusion

In this study, we examined and captured six MDE settings within SAP in detail, which gave us some general insights in the usage of MDE techniques during software development within SAP. Further, we got insights in the motivation for usage of different MDE techniques.

The study gave us a first idea how an MDE setting influences on a fine granular layer the process of developing software. We compared addressed internal software quality attributes with the MDE settings structure to identify whether current theoretical explanations for MDE influences are sufficient. Thereby, we identified that there is still a lack of knowledge concerning the degree of the influence of a single MDE technique to internal software quality attributes. Thus, it is even harder to evaluate the influence of a whole MDE setting where multiple MDE techniques are used in combination.

The diversity of the captured MDE settings, point out that capturing them explicitly can be crucial in future. This is not only due to the need to train new developers in MDE settings. Further, MDE settings might also include critical constellations that decrease productivity, or especially changeability. Evolution of MDE settings contains therefore always the risk to lead to disadvantageous constellations. Capturing MDE setting explicitly, will be the basis for analysis of these risks. This is especially important if different parts of a setting are applied by different groups or units of developers, which might otherwise be not aware how changes in their part of the MDE setting influence parts applied by other groups. In addition, capturing MDE settings explicitly, can help to identify evolution tendencies early on, which enables planning. Finally, MDE settings present also how development depends on different technologies, such as tools, or languages. Capturing how tools are used in combination within an MDE setting, can be important to identify risks such as technology login early on.

In long term, research should lead to a better understanding how MDE settings influence productivity. It is a goal to provide techniques that support identification of risks, e.g. for the changeability, and resolution approaches. This should support strategic decisions concerning evolution of MDE settings.

Acknowledgments

We are very grateful for the cooperation of our contact persons and interview partners at SAP AG, namely Andreas Bold, Hilmar Demant, Aalbert de Niet, Mario Herger, Jens Hertweck, Frank Jentsch, Andreas Krompholz, Stefan Schreck, Florian Stallmann, Cafer Tosun, Axel Uhl, Bertram Vielsack, and Carsten Ziegler as well as our interview partner Marcel Salein at FIVE1 GmbH & Co. KG. Further, we thank Gregor Gabrysiak for feedback and discussions on the design of the study.

References

[1] R. Agarwal and M. R. Tanniru. Knowledge acquisition using structured interviewing: an empirical investigation. *J. Manage. Inf. Syst.*, 7:123–140, June 1990.

[2] Helmut Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung*. Spektrum, 1996.

[3] Helmut Balzert. *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum, Heidelberg, 1998.

[4] Barry W. Boehm. Improving Software Productivity. *Computer*, 20:43–57, September 1987.

[5] Carsten Bönnen and Mario Herger. *SAP NetWeaver Visual Composer*. SAP PRESS, 2007.

[6] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, World Wide Web Consortium (W3C), 2001.

[7] Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13:3–21, 1990. 10.1007/BF00988593.

[8] Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 176–185, Washington, DC, USA, 2006. IEEE Computer Society.

[9] 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, 1 edition, 2003.

[10] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley, October 2010.

[11] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[12] Volker Gruhn, Daniel Pieper, and Carsten Röttgers. *MDA: Effektives Softwareengineering mit UML2 und Eclipse*. Springer, Berlin, 1 edition, 2006.

[13] Werner Heijstek and Michel R. V. Chaudron. Empirical Investigations of Model Size, Complexity and Effort in a Large Scale, Distributed Model Driven Development Process. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '09, pages 113–120, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Werner Heijstek and Michel R.V. Chaudron. The Impact of Model Driven Development on the Software Architecture Process. *Software Engineering and Advanced Applications, Euromicro Conference*, pages 333–341, 2010.

[15] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 633–642, Waikiki, Honolulu, HI, USA, 2011. ACM.

[16] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[17] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, "Hoboken, NJ", 2008.

[18] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, Boston, MA, USA, 2003.

[19] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[20] Katrina D. Maxwell, Luk Van Wassenhove, and Soumitra Dutta. Software Development Productivity of European Space, Military, and Industrial Applications. *IEEE Trans. Softw. Eng.*, 22:706–718, October 1996.

[21] Niklas Mellegaard and Miroslaw Staron. Characterizing model usage in embedded software engineering: a case study. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 245–252, New York, NY, USA, 2010. ACM.

[22] Parastoo Mohagheghi and Vegard Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, pages 432–443, Berlin, Heidelberg, 2008. Springer-Verlag.

[23] Parastoo Mohagheghi, Miguel A. Fernandez, Juan A. Martell, Mathias Fritzsche, and Wasif Gilani. Models in Software Engineering. chapter MDE Adoption in Industry: Challenges and Success Criteria, pages 54–59. Springer-Verlag, Berlin, Heidelberg, 2009.

[24] Object Management Group. *MDA Guide Version 1.0*, May 2003. Document omg/2003-05-01.

[25] OMG. UML 2.0 Superstructure Specification, Object Management Group, Version 2.0, formal/05-07-04, 2005.

[26] Harry M. Sneed. *Softwaremanagement*. Verlagsgesellschaft Rudolf Müller, 1987.

[27] Thomas Stahl, Markus Völter, Sven Efftinge, and Arno Haase. *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*. dpunkt Verlag, 2 edition, 2007.

[28] Thomas Weigert, Frank Weil, Kevin Marth, Paul Baker, Clive Jervis, Paul Dietz, Yexuan Gui, Aswin Van Den Berg, Kim Fleer, David Nelson, Michael Wells, and Brian Mastenbrook. Experiences in deploying model-driven engineering. In *Proceedings of the 13th international SDL Forum conference on Design for dependable systems*, SDL'07, pages 35–53, Berlin, Heidelberg, 2007. Springer-Verlag.

[29] Jon Whittle and John Hutchinson. Empirical Assessment of the Efficacy of MDE. http://www.comp.lancs.ac.uk/ eamde.

A Examined MDE Settings

This Appendix includes the MDE settings captured in this study. Beforehand, we give a short introduction into the notation used in this appendix. The notation bases on UML activity diagrams [25]. We first use activity diagrams to illustrate the fine granular process. In addition we used three additional notation elements, illustrated in Figure 6. First, we use organizational units, to indicate that a specific devision is responsible for certain activities. Further, we use additional entry and exit points notated as blue initial node respectively final node. These additional entry and exit points indicate that developers might choose to repeat or redo certain development steps.

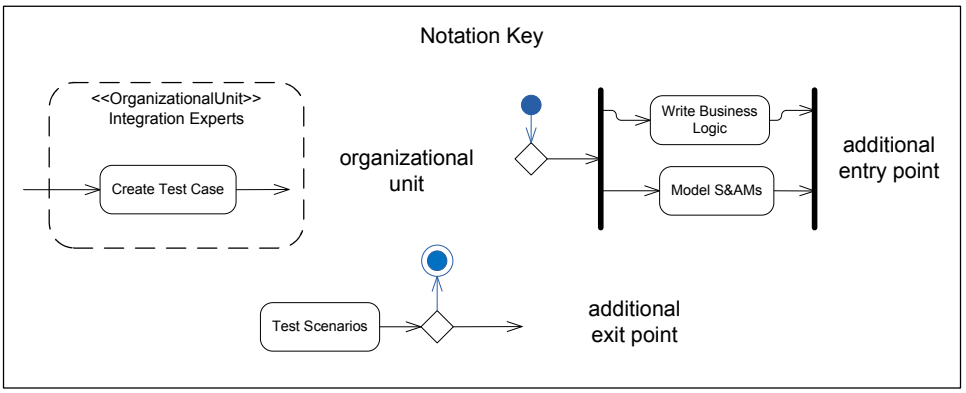

Figure 6: Additional notation elements for activity diagram

In addition to the fine granular process, we illustrate more information about single activities in a detailed activity view. Therefore we use further notation elements, introduced in Figure 7. These elements indicate, whether the activity is performed manually, semi-automatically or automatically (1-3). Further, one or more alternative tools supporting the activity can be annotated (4 and 5). Artifacts that are input and output of an activity (6 and 7) can be annotated with a multiplicity indicating how many artifacts are input or output (8). Artifacts can also be part of other artifacts (9). Further, we sometimes indicated the language of an artifact using a conforms-to relation (10). We also sometimes indicate a similarity in roles of the artifacts with an 'inheritance' (11). Finally, some activities are composed of smaller activities (12).

A.1 Business Object Development at FP 2.0 (BO)

SAP Business ByDesign is build out of deployment units containing different process components, that are necessary to implement the desired business processes. The process components are the development units in ByDesign. The name spaces are cut along the lines of the PC's. Thereby the desired functionality is build in terms of so called business objects, that are part of process components. If process components have to interact across the borders of the deployment units, process interaction models are used to describe the interaction. In this report we do not capture the MDE setting for the current state of development of business objects. Instead, this report aims to capture the development of business objects for the product version FP2.0. This MDE setting contains work with models and generation of code out of models and, thus, is clearly a model driven approach.

The order of activities of the MDE setting for the business object development at FP2.0 is shown in Figure 8 in form of a slightly adapted activity diagram. Thereby, the activity diagram is split into

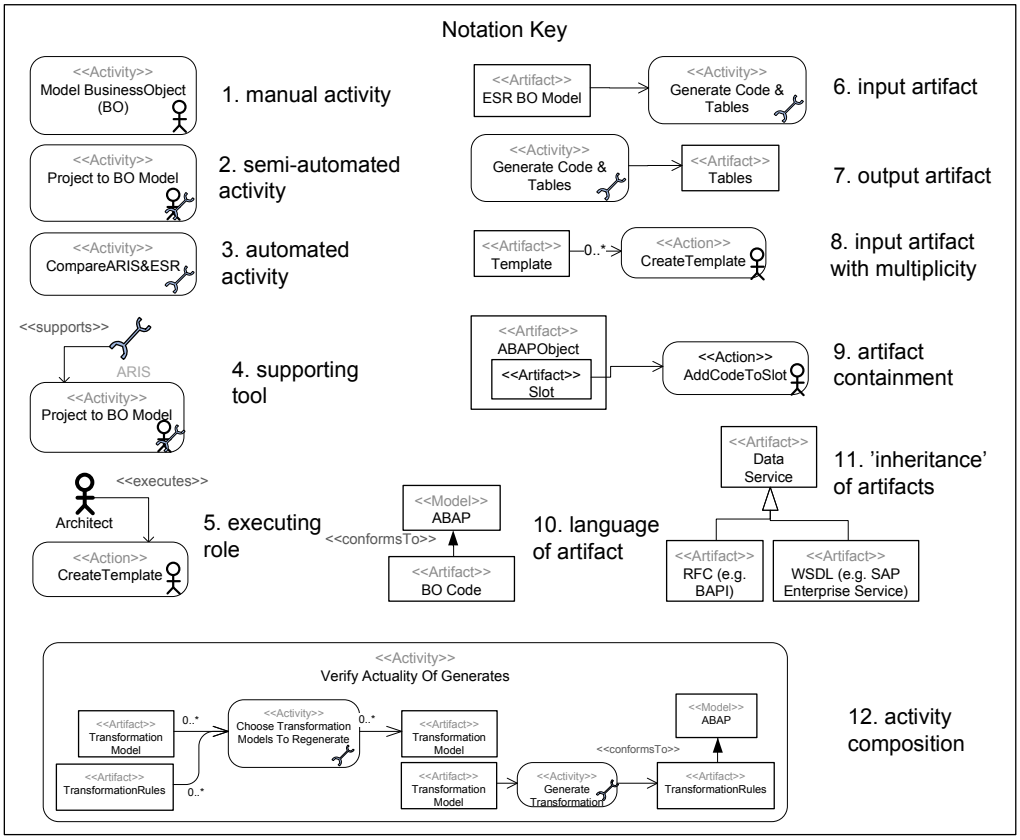

Figure 7: Notation elements for detailed activity view

four parts, which indicates that the activities are applied by different organizational units. Most activities are applied by the business object development team. However, as the development of a business object might start on the basis of an integration scenario it is necessary to verify that the process component containing the business object fits the integration requirements. Thus, the integration experts apply tests to the process components containing the developed business objects. Further, the solution management retrieves initially the requirement. Finally, the test department tests the resulting functionality against the requirements.

In addition, we annotated additional initial nodes and flow final nodes in the diagram (marked in blue). We use this to indicate that the developers might stop the process, e.g. due to a failed test, and restart the process at another activity to apply changes or correct something. For example, if the activity test scenarios uncovers a fault, this might lead to the need for further changes in the business objects and thus the process might restart with the activity write business object logic. However, another reasonable reaction might be to repeat the process starting with the activity adapt business object model.

Further, the second entry point at the activity create ESR model indicates that, e.g. for the development of prototypes, the developer might directly start the implementation in ESR. Thereby, they should, however, go back in the development process and apply the modeling activities in ARIS afterwards. Thereby, it might be necessary to correct the ESR implementation.

Figure 9 shows the activity define requirements with customer in detail. Thereby the solution management defines with with the help of the customer the requirements. The stick man indicates

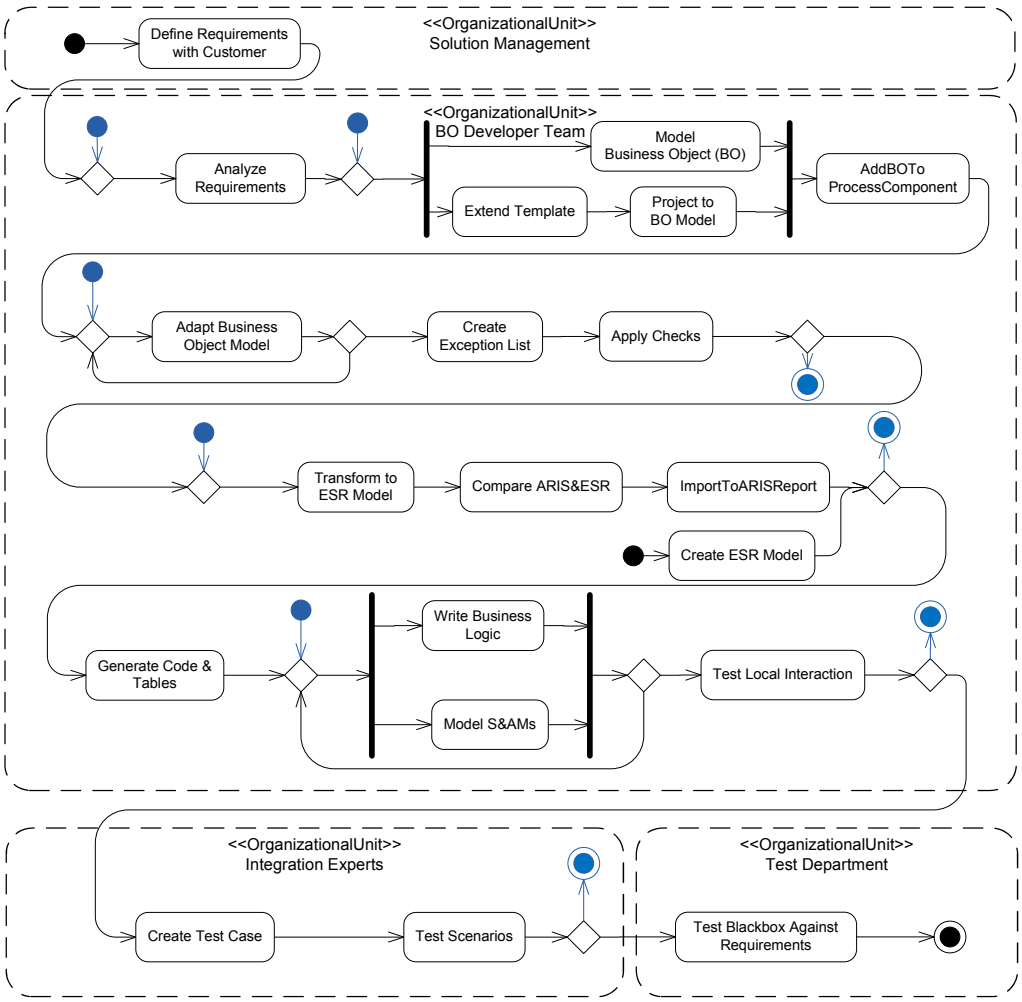

Figure 8: BO: Order of activities in MDE setting for the Business Object Development at FP 2.0

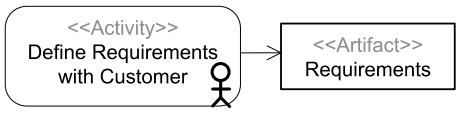

Figure 9: BO: Activity 'Define Requirements with Customer'

that the activity is performed manually.

Figure 10 shows the activities analyze requirements, model business object, extend template, project to BO model, add BO to process component, and adapt business object model more in detail. Thereby, it is indicated by the stick man that all activities are manual tasks. Further, most of the activities are supported by the tool ARIS.

The figure shows, which artifacts are input and output of the activities. Thus the activity analyze requirements starts on the basis of the requirements defined by the solution management. As result a specification for the business object is built.

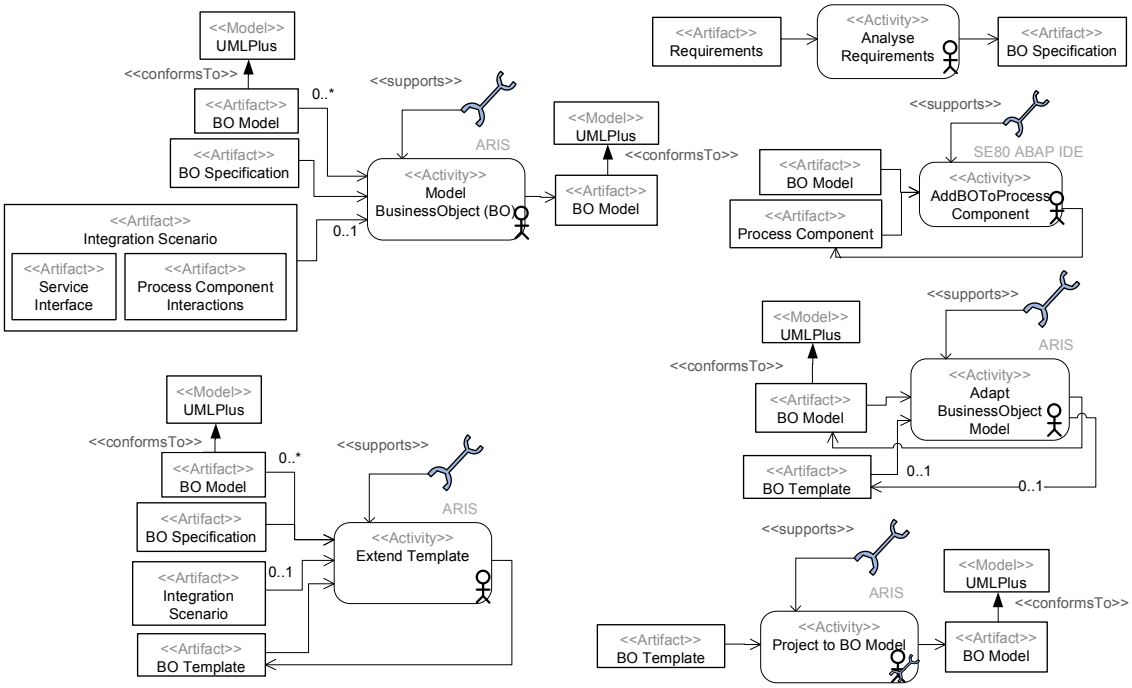

Figure 10: BO: Activities of MDE setting for the Business Object Development at FP 2.0

For modeling the business object the developer might use different artifacts, such as BO Models of business objects that are used by the business object currently modeled. The BO Specification contains besides the specification of the BO itself, also a data model. Finally, an Integration Scenario can be input. Thereby, an Integration Scenario describes which process components interact and via which service interfaces they interact. This is important since the business object implements a part of a process component and might therefore be responsible for or depend on the described interactions.

Alternatively, the developers might use a BO Template for the creation of the BO Model. In this case they have to extend the template first. This happens manually with the activity 'Extend Template'. Similar to 'Model Business Object', 'Extend Template' can be performed on the basis of BO Specification and Integration Scenario and can lead to links to existing business objects.

After the template was extended the BO Model is created as a projection of the template. This is done semi-automatically by the activity 'Project to BO Model'. Thereby the developer has to select the nodes, relations and data types that have to the projected.

The BO Code has to be added manually to the Process Component. For the implementation of a process component it is further necessary to implement process agents although this is not part of the MDE setting for the development of a business object.

Finally, the activity 'Adapt Business Object Model' is used to manipulate the BO Model further. If a template was used for the creation of the BO Model, this activity can also change the template.

The activities, shown in Figure 11 more in detail, are used to apply checks on the BO Model in ARIS. Thereby, the used checks are defined commonly for specific business object types. In addition, the developer can create an exception list that defines how false positives can be excluded from the test result.

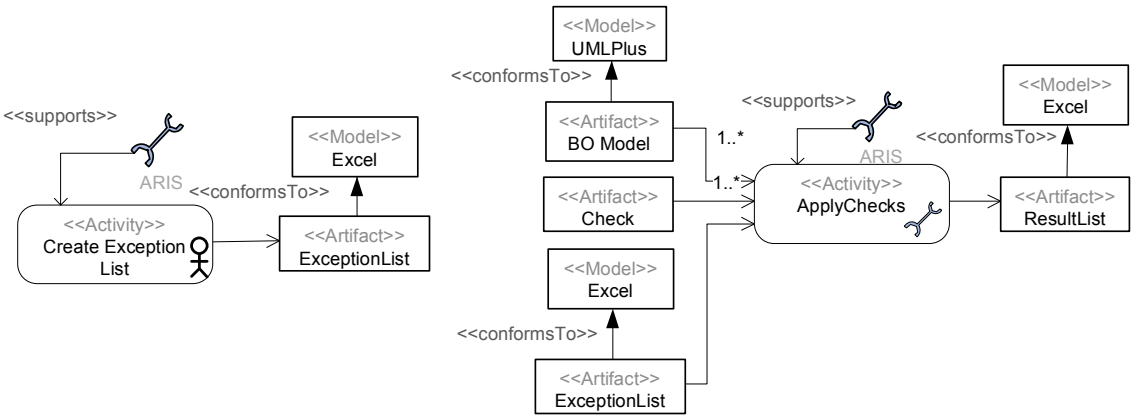

Figure 11: BO: Activities of MDE setting for the Business Object Development at FP 2.0

With the activity 'Apply Checks' the defined checks can be automatically applied in ARIS on a given set of business objects. As result an Excel list of check failures is created.

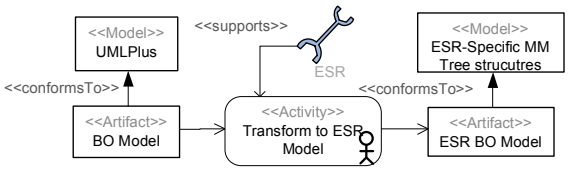

Figure 12: BO: Activities of MDE setting for the Business Object Development at FP 2.0

After the BO Model is created and checked, the model has to be transported from ARIS to ESR. It is required to transform the BO Model to an ESR specific model. This is done manually by the activity 'Transform to ESR Model'. The activity is shown in Figure 12.

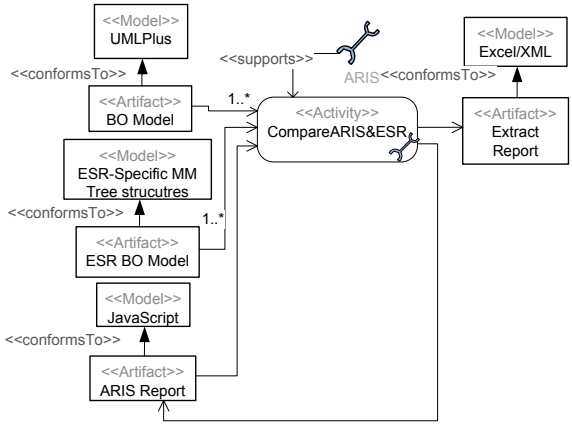

Figure 13: BO: Activity 'Compare ARIS & ESR'

Since the transformation between the BO Model in ARIS and the ESR BO Model in ESR is done manually, it is necessary to compare the models. The activities necessary for that are shown in Figure 13. The comparison is done automatically by the activity 'Compare ARIS & ESR', which is

supported by ARIS. The result of the comparison is written to an Excel/XML file which is further added to the ARIS Report.

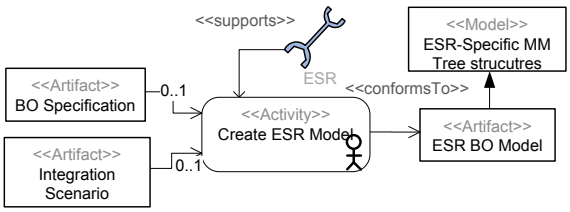

Figure 14: BO: Alternative start activity, for prototype implementations.

The activity 'Create ESR Model' is applied when the development it for some reason not started with the modeling in ARIS. Thereby, the developer start with creating the ESR Model. This might base on the BO Specification and the Integration Scenario, too.

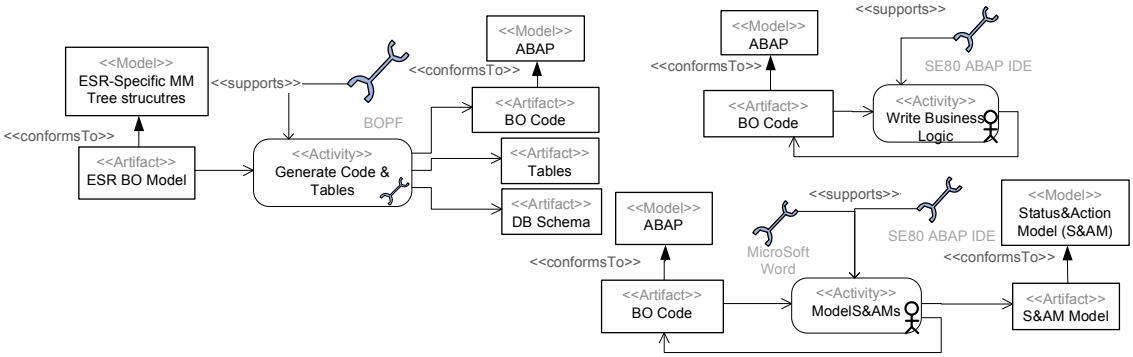

Figure 15: BO: Activities of MDE setting for the Business Object Development at FP 2.0

Figure 15 shows the activities deal with the implementation of the business object. With the help of the tool BOPF the activity 'Generate Code & Tables' automatically generates, based on the ESR BO Model, a data base schema, tables and BO Code in ABAP. This code can now be further implemented. Therefore, the developer can just extend the BO Code manually within the SE80 ABAP IDE. Alternatively, the developer can manually create Status & Action Models (S&AMs) using MicroSoft Word and reference them by the BO Code using the SE80 ABAP IDE (activity 'Model S&AMs'). An S&AM describes behavior and can be interpreted during the execution.

Figure 16 shows activities that are used to test the developed business object. First, the developers of the business object test it manually considering the BO Code, the S&AM Models, the DB schema and the tables (activity 'Test Local Interaction'). If the business objects was developed considering an integration scenario, the developers hand the responsibility over to the integration experts. They create manually test cases based on the integration scenario (activity 'Create Test Case'). Based on that test cases the process components that base on the created business objects are tested.

Finally, the test department applies manually blackbox tests on the process components to validate the behavior against the requirements (activity 'Test Blackbox Against Requirements').

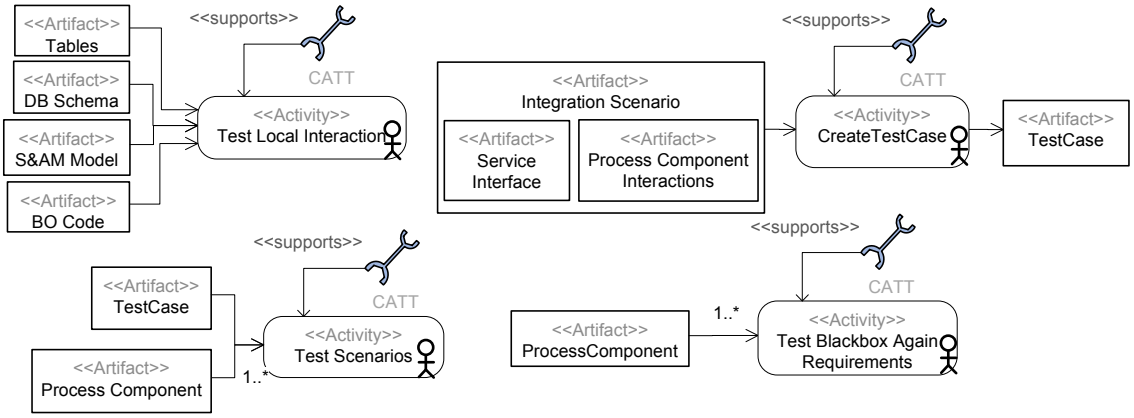

Figure 16: BO: Activities of MDE setting for the Business Object Development at FP 2.0

A.2 Service Implementation Workbench (SIW)

The Service Implementation Workbench can be used to generate artifacts that provide access to further functionality. Thereby, SIW is often used to generate a web service that provides access to ABAP functionality. There are three scenarios for building a web service with the help of the SIW.

In the Proxy-based case the development starts with a web service description, a proxy and ABAP code with a BOR/Bapi signature. The result is a web service that fulfills the given web service signature and provides the functionality that it accesses via the BOR/Bapi interface.

In the API-based case the development starts only with the ABAP code and the corresponding BOR/Bapi signature.

Finally, it is possible to use the SIW without web service signature and ABAP code with corresponding BOR/Bapi signature ('service independent development'). In that case no service is created. The created functionality might for example be accessed via RFC (remote function call). Thereby, it is necessary to use templates that contain much more information than templates for the Proxy-based or API-based case.

An MDE setting that is also supported by the SIW describes how templates and configurations are build for the SIW. Finally, this report includes the MDE setting for testing the services created with the SIW. Although these MDE settings are not directly supported by the SIW they are included here.

SIW enables the use of templates and the generation of code. Since web service interface description, templates and code can be seen as artifacts/models we consider MDE settings for the web service development with the SIW to be MDE approaches.

This section describes the MDE settings that are supported by the service implementation workbench. The MDE settings of proxy-based case (Section A.2.1), API-based case (Section A.2.2), and the case without given signature (Section A.2.3) describe activities that are applied during implementation phase of a software development process, but include partly also quality assurance activities.

The MDE setting of the configuration creation cannot easily be assigned to a software development process activity if the development of the service is considered. However, it might be seen

as a part of the 'tool development' where it would belong to the implementation phase, too.

The MDE setting for testing the developed service can be assigned, for example to the 'develop and verify next level product' phase of a spiral model, the construction and transition phases of a RUP development process or to the Unit Test phase in a eXtreme Programming development process.

A.2.1 The proxy-based case

Figure 17 shows order of activities of the MDE setting of the proxy-based case. The Figures 18 and 19 show the activities of the MDE setting of the proxy-based case. In proxy-based case the development starts with a web service signature (formulated with WSDL) and a proxy. If this proxy does not exist it might be generated with the help of the tool Transaction SPROXY. Thereby, the proxy class, the interface of the proxy and types and structures for the data dictionary are created, based on the service signature and a corresponding XSD. Proxy class and interface are formulated in ABAP. If the service signature changes during development the proxy can be regenerated. Thereby, the manually implemented parts of the proxy are preserved.

At the start the developer has to choose an appropriate configuration and with it the templates that are used during the development. Based on the chosen configuration the SIW project can be created. The based on the resulting project meta data the developer has to set certain context parameters that are relevant for the later generation steps. All three activities (choose configuration, create SIW project and set parameters) are manual tasks.

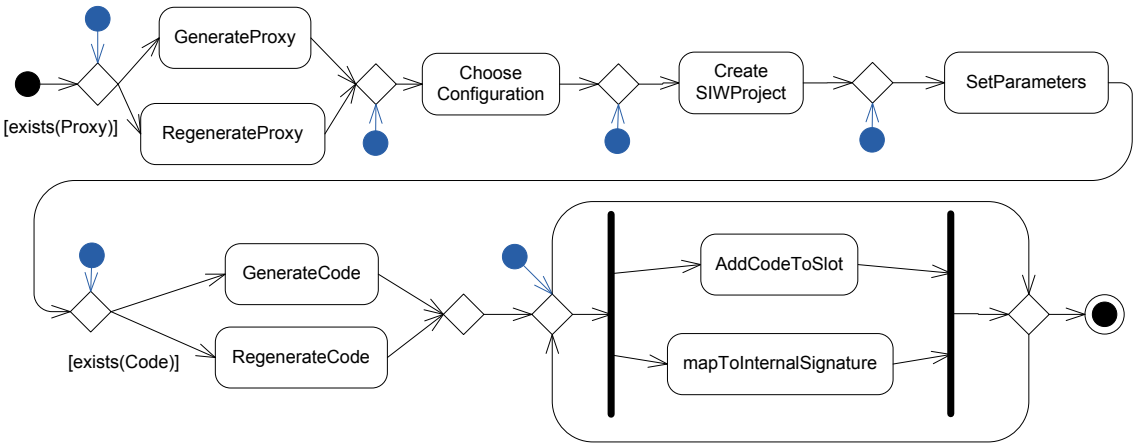

Figure 17: SIW: Activity diagram of the order of activities in the MDE setting of building a service in the proxy-based case

After the configuration is chosen and the context parameters are given by the developer the code of the service can be generated. Input for this generation are the configuration, templates of the configuration, context parameters and the proxy class. Results of the fully automated generation step may be:

- a set of entries in the data dictionary, such as data elements, structures, table types, or domains
- a set of ABAP objects, such as classes, interfaces, method implementations, programs, class includes, functions, or modules

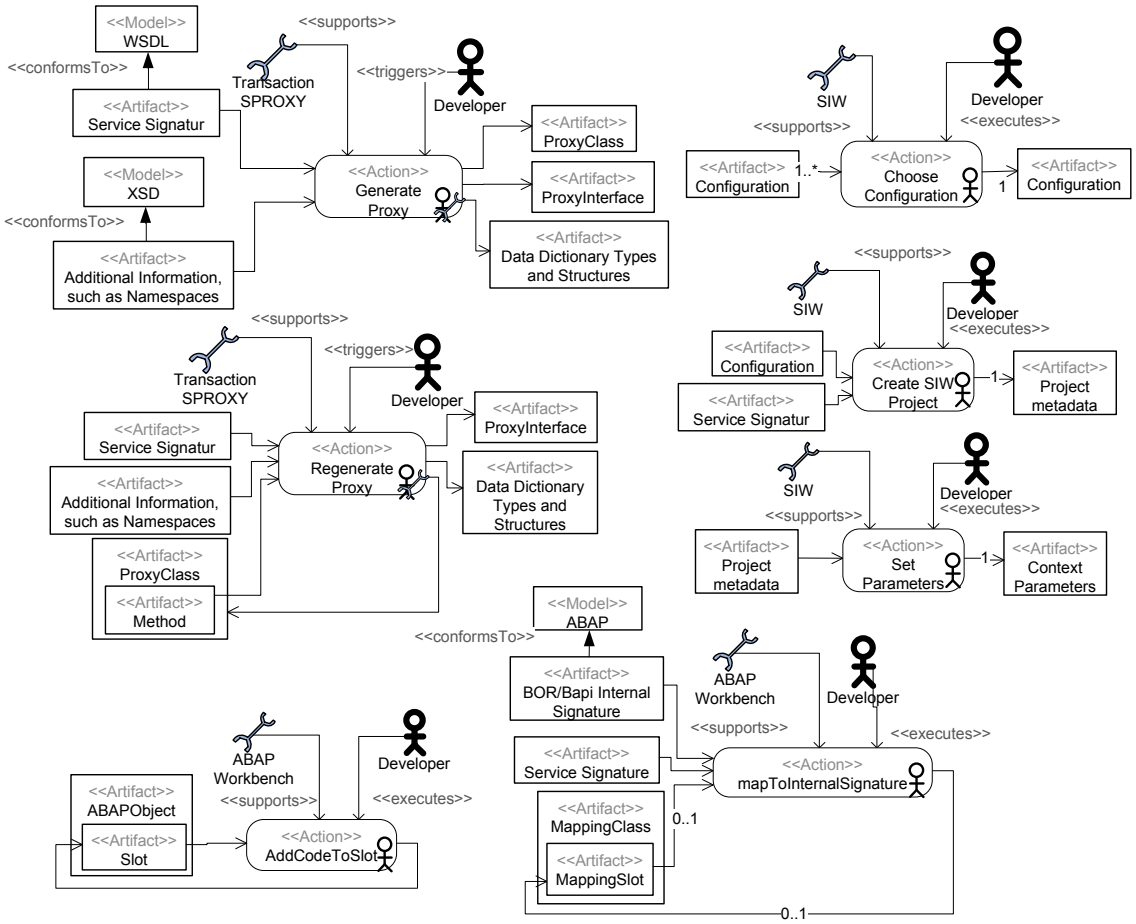

Figure 18: SIW: Activities of the MDE setting of building a service in the proxy-based case

- further, objects, such as table entries, registries, or service groups
- a mapping class that contains static parts but also code slots that should later be filled with code manually
- and method implementations for the proxy class.

If later in the development process context parameters or parts of the proxy change not all these artifacts have to be regenerated. Here SIW supports synchronization. That means, only the static parts of the mapping class and the ABAP objects have to be regenerated. The code slots remain untouched. Further, SIW is able to only regenerate objects that will change. Therefore, SIW generates the code temporary and compares it with the old code. Only if there are changes an object is replaced. This saves performance of development, since a full generation of objects requires persistence, which means additional effort, such as instantiation and applicaiton of transport jobs.

The generation of the code is not the last step in the MDE setting. There are two further activities. First, the developer can manually add code to the generated code slot within the ABAP objects. Second, the developer has to complete the mapping to the BOR/Bapi signature. Based on the BOR/Bapi signature and the service signature, which was initially used to generate the proxy, the developer has to implement the mapping by filling the mapping slots within the generated mapping class. This is a manual activity.

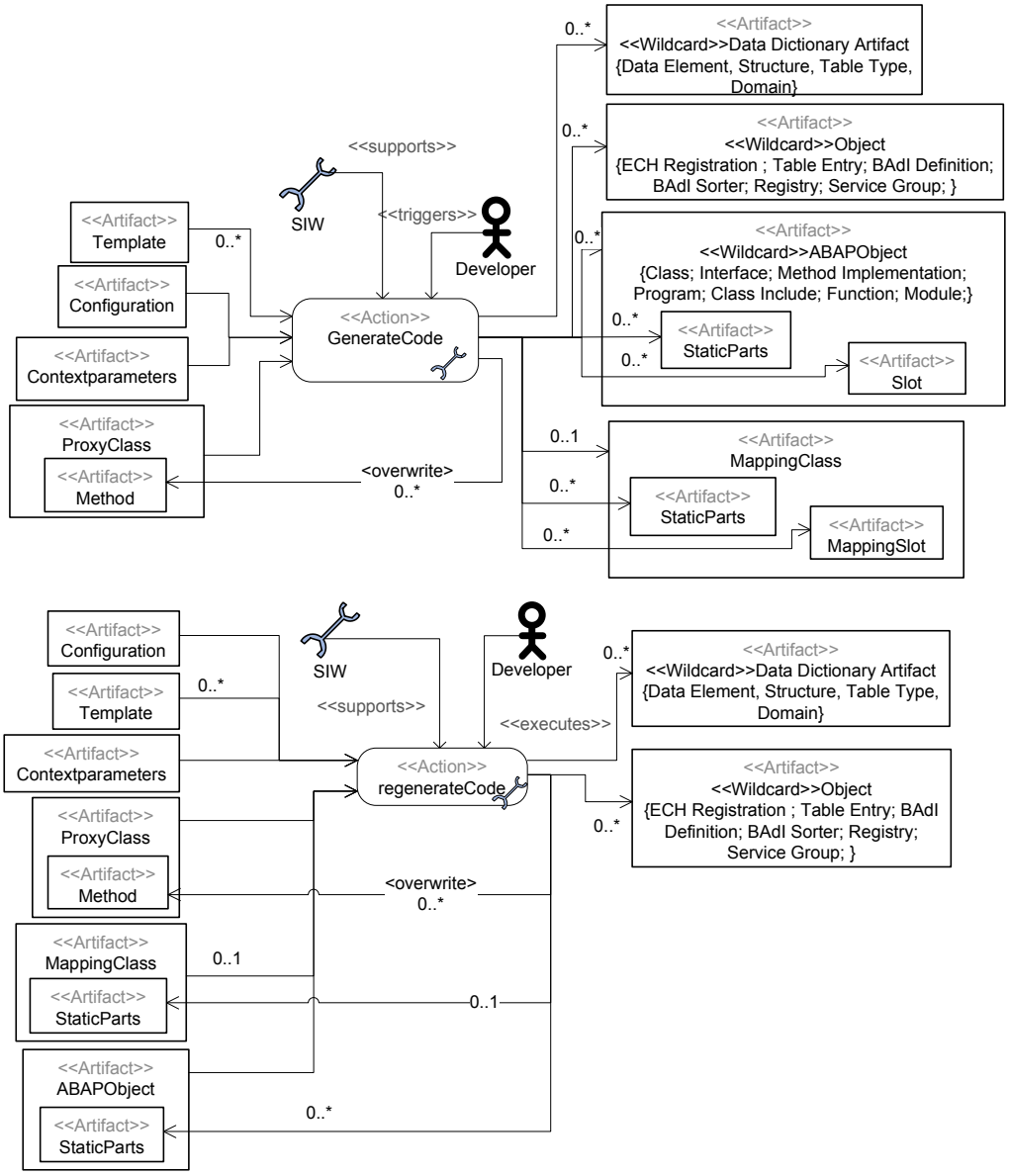

Figure 19: SIW: 'Generate code' and 'regenerate code' activities of the MDE setting of building a service in the proxy-based case

Within the MDE setting shown in Figure 17 some additional start activities are marked blue. During development a developer might decide to go back and repeat some of the development steps (iteration). The additional start activities should imply, where the developer might start again. For example, after a change in the provided WSDL signature it might be necessary to generate the proxy again. A developer might also redo the configuration choice or reset the context parameters. If context parameters change also the code generation might be repeated. Finally, the developer might always decide to change the implementations in the code slots and the mapping slots.

A.2.2 The API-based case

The API-based case differs from the proxy-based case, since no web service interface and proxy are given. Figure 20 shows the order of activities of the MDE setting of the API-based case. The Figures 21, 22 and 23 show the activities of the MDE setting of the API-based case.

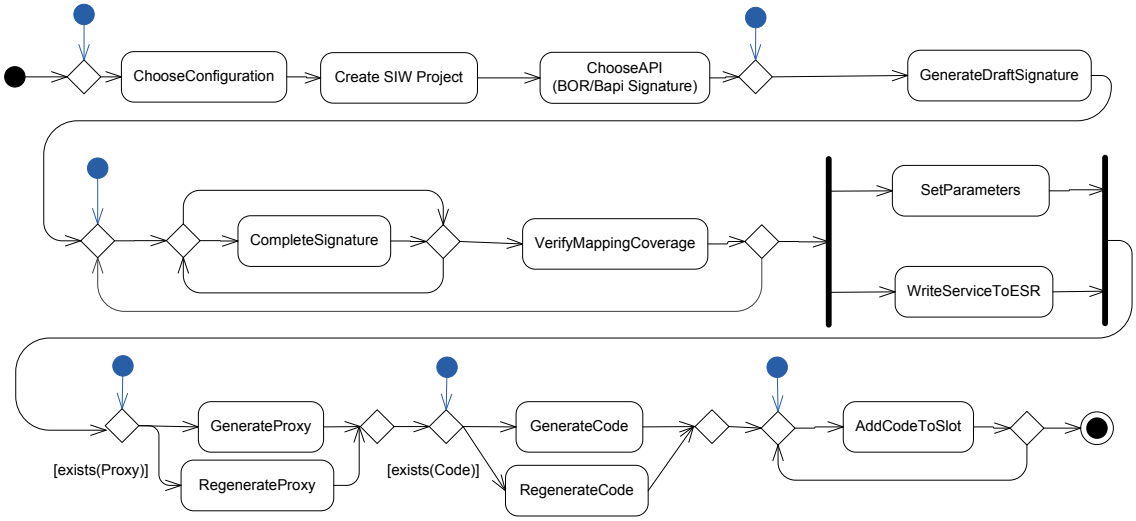

Figure 20: SIW: Activity diagram of the order of activities in the MDE setting of building a service in the API-based case

The first step in this case is to choose the configuration, create the SIW project, and choosing the API. Then the web service signature and proxy are retrieved. Thereby, first a draft signature is automatically generated out of the BOR/Bapi signature and the configuration. This is implemented by a populator, which is specified in the configuration. The created signature is not a WSDL signature but a SXF (Signature Exchange Formate) signature. The SXF signature is used to handle the mapping information between internal and external signature, as long as the proxy is not yet generated. The populator might not generate a complete signature. In this case the user might manually complete the signature.

Then the MDE setting includes an automated verification activity, to ensure a full coverage of the mapping between service signature and BOR/Bapi signature (action 'verify mapping coverage'). If this verification fails the signature can be further manipulated. Next the SXF signature is transformed automatically to a WSDL web service signature, which is written to ESR. Parallel to that, the context parameters can be set. Finally, the proxy (proxy class, proxy interface, and types and structures for the data dictionary) is created based on the WSDL service signature and the corresponding XSD. This is done semi-automatically. If the signature is changed later in development, the proxy might be generated again. Parts of the proxy that are filled during the code generation step are preserved by regenerating the proxy.

The generation of the code is equal to the proxy-based case, but it does not lead to a creation of a mapping class that has to be filed manually. According to that also a regeneration changes from the proxy-based case. As in the proxy-based case, slots of ABAP Objects have to be filled with code manually.

Also the MDE setting in Figure 20 includes additional start activities. In addition there is a flow between the end of the verification of the mapping coverage and the point where the draft signature and the mapping can be completed manually. This should indicate that a fail of the verification, causes the developer to go back to that point in the MDE setting.

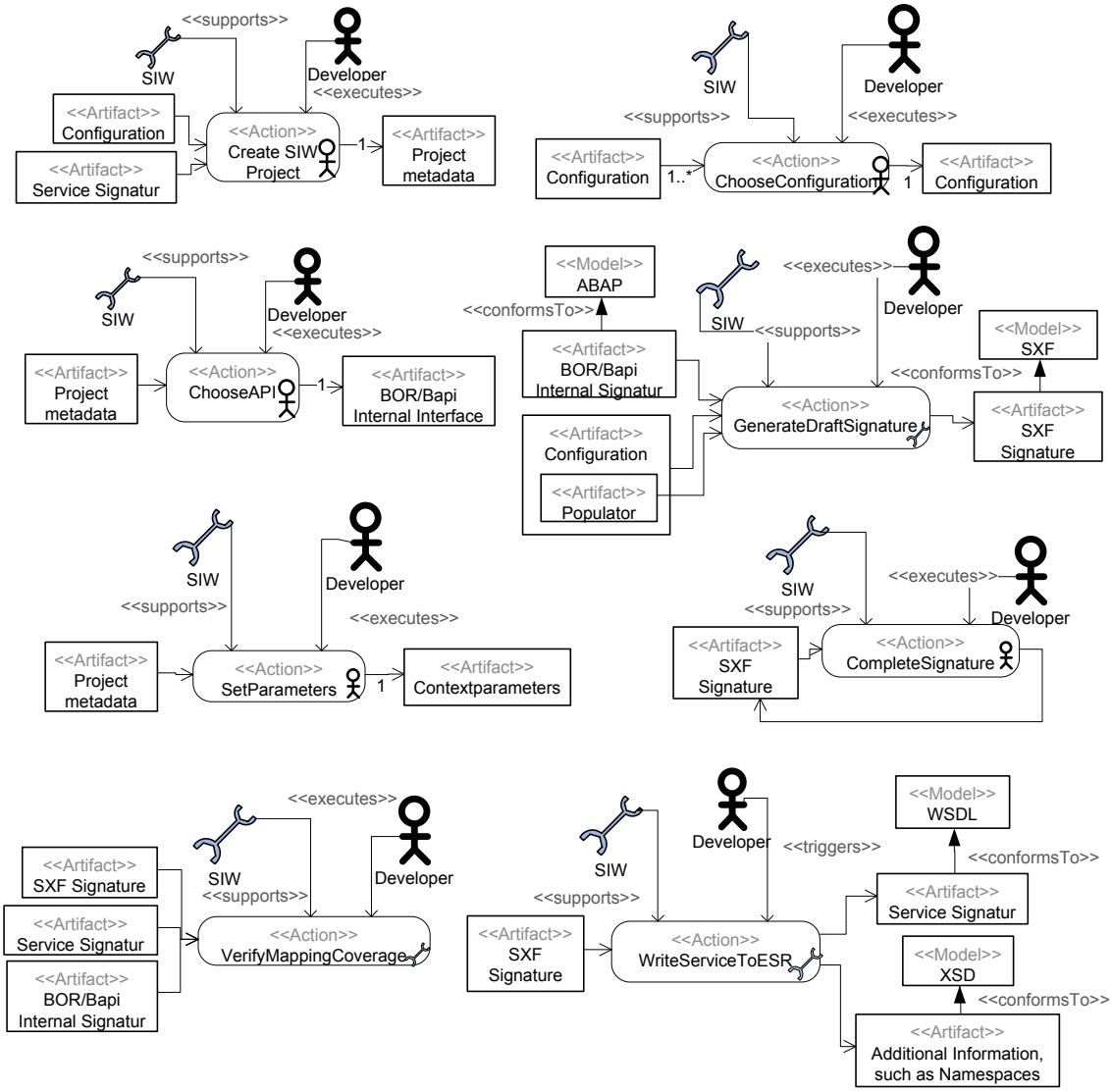

Figure 21: SIW: Activities of the MDE setting of building a service in the API-based case

A.2.3 Case of service independent development

Figure 24 shows the order of activities in the MDE setting of the service independent development. The Figure 25 shows the activities of the MDE setting of the service independent development. In case no signature is given, the MDE setting becomes quite simple, since no proxy and no mapping between interfaces are required.

Like in the other cases, the developer has to choose a configuration, create the SIW project, and set context parameters, first. Then it is directly possible to generate code based on the configuration, the associated templates and the context parameters. In contrast to the API-based case and the proxy-based case the generation does neither lead to changes in some proxy class nor to the creation of a mapping class. If context parameters change later, the generation can be redone, with preservation of the code in the slots of the ABAP objects. The code in the slots of ABAP objects is added manually after the generation activity.

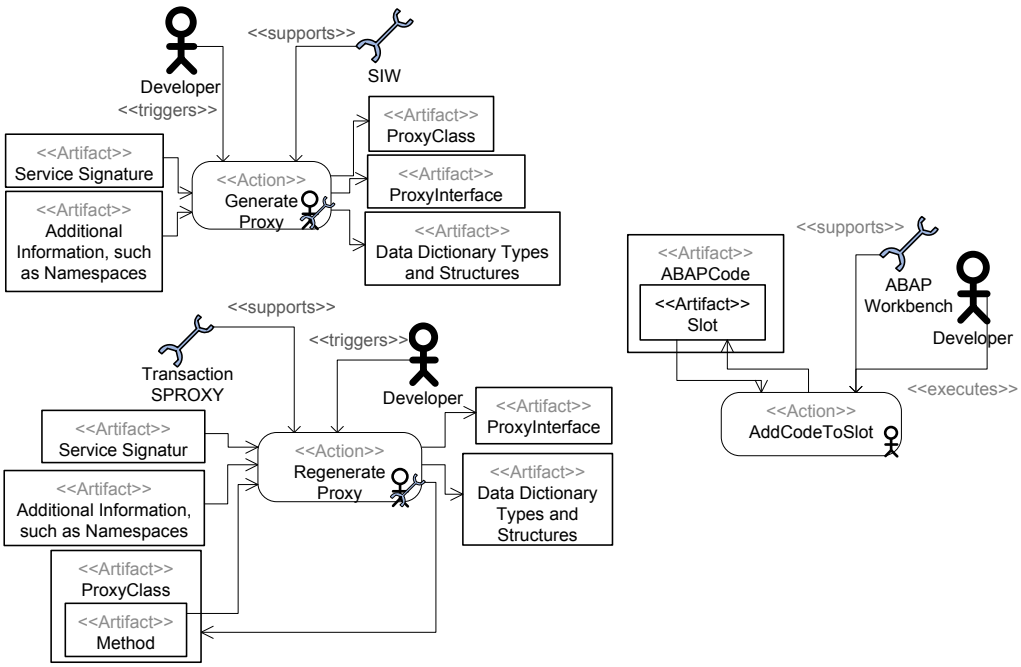

Figure 22: SIW: Activities of the MDE setting of building a service in the API-based case

A.2.4 Creation of a configuration with templates

Figure 26 shows the MDE setting for building a configuration with templates within the SIW. First, templates have to be created. Thereby, multiple other templates might be used as samples. The resulting template conforms to a report-include of the ABAP workbench SE38. After the necessary templates are created, the configuration can be created based on the templates. Thereby, other configurations might be used as samples, too. The additional start activity implies that it is also possible repeat the creation of a configuration.

Finally, a configuration might include a populator or reader. Like the creation of a template populators and readers might be created based on already existing populators and readers.

All activities are manual activities and supported by the SIW. In contrast to the service creation the creation of a configuration is not performed by developers but by architects.

A.2.5 Testing the created service

Figure 27 shows a MDE setting for testing the created services. The SIW supports the developer in manually creating and storing test files formulated in XML.

There are several alternatives for testing. Using tools like the WSNavigator the developer can manually test the service. Automatic tests are possible with the tools soapUI and eCATT. The tool soap UI requires in addition to the service signature and the test XML information about the port, where the service can be accessed. The tool eCATT can be used to formulate post conditions for the test and to automatically execute the tests.

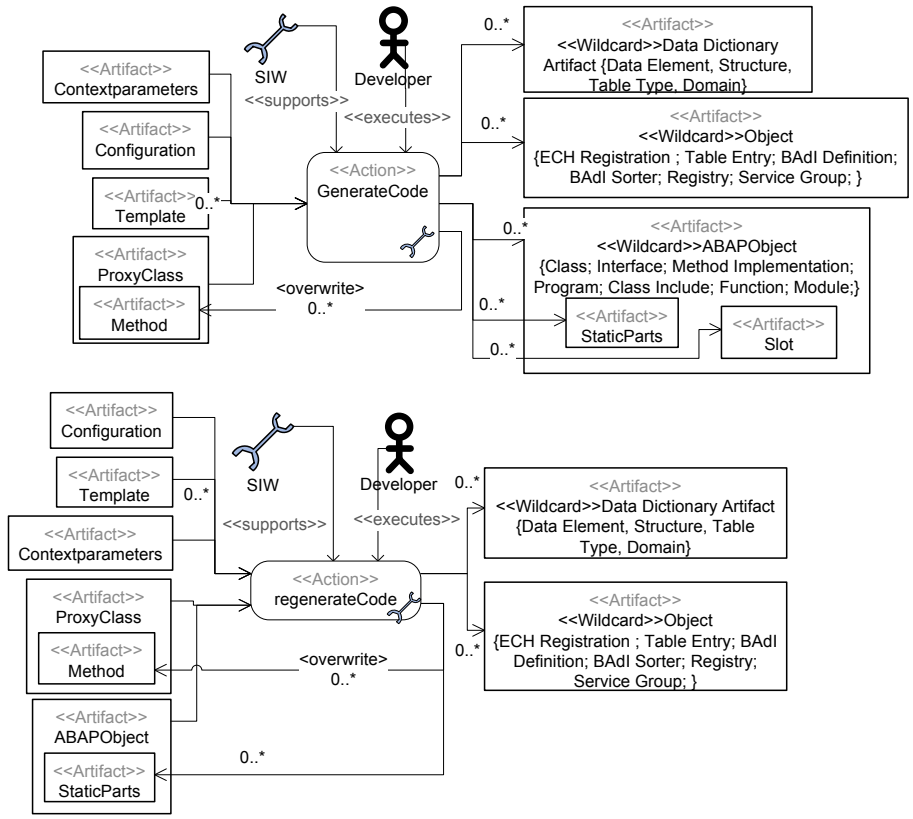

Figure 23: SIW: Activities of the MDE setting of building a service in the API-based case

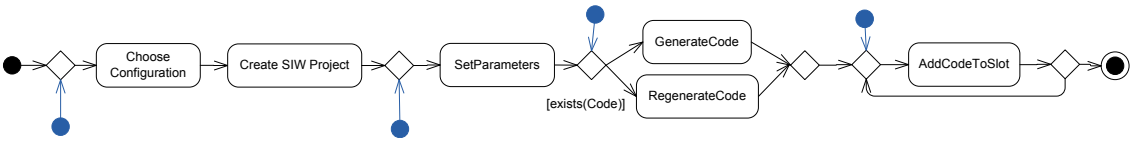

Figure 24: SIW: Activity diagram of the activities in the MDE setting in case of service independent development

A.3 Visual Composer (VC)

Visual Composer (VC) is a tool for the model based creation of user interfaces for Net-Weaver applications. Thereby, VC allows the user to define the data flow between UI elements and data services. Further, the layout of the UI elements can be defined. Thereby, the user defines data flow and layout completely via models. The resulting models are then compiled and deployed to the required runtime.

In the following we want to introduce the MDE setting that can be used to create an application using Visual Composer. Thereby, this report does not contain all possible ways, but a set of samples.

For the creation of an application using Visual Composer a set of activities is necessary. First, the deployment component and the model have to be created. Then it is necessary to integrate

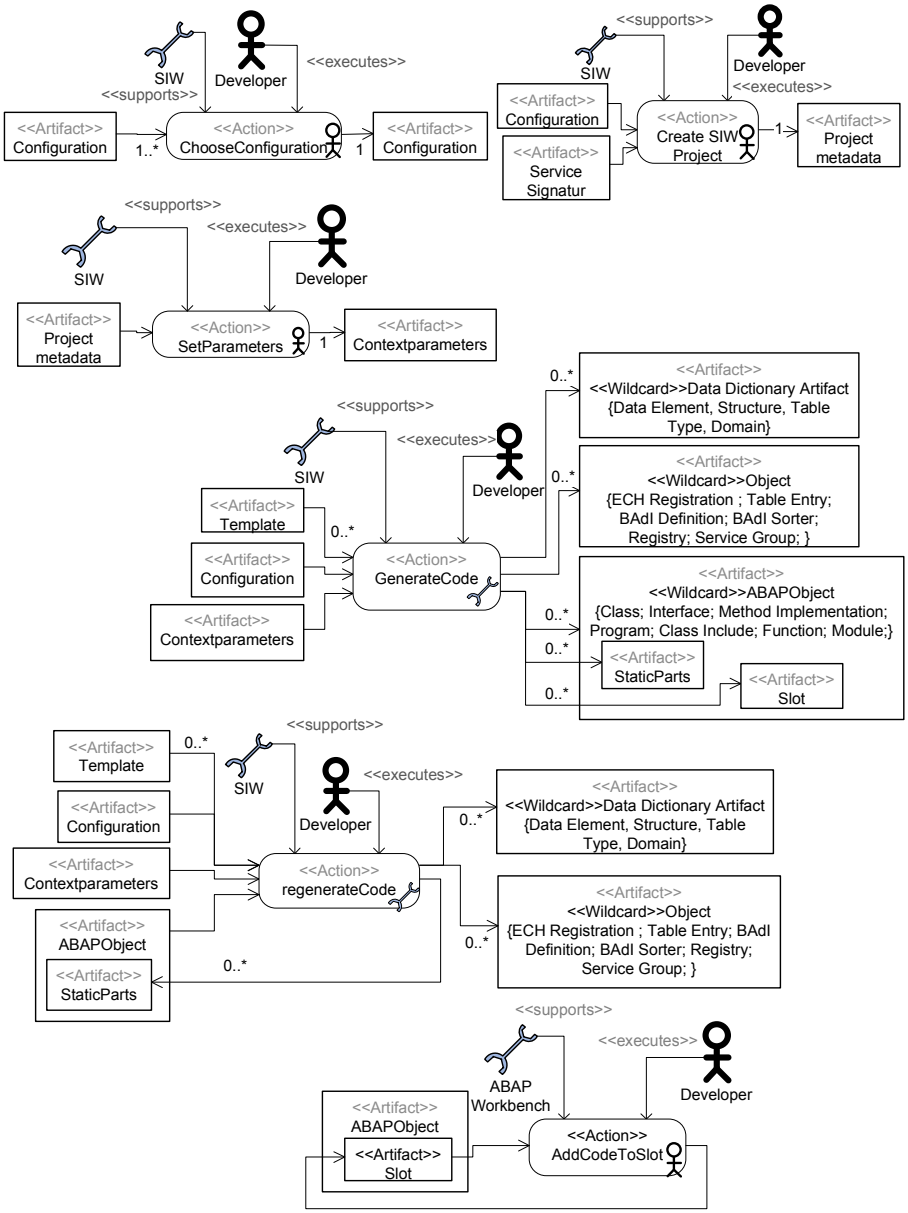

Figure 25: SIW: Activities of the MDE setting in case of service independent development

data services such that application logic and informations can be build into the application. Then model elements for the UI have to be connected to these data services and further have to be manipulated to reach a desired layout. Finally, the model has to be deployed. The integration of data services, the connection to UI elements, and the layout activities do not need to be executed at once, but can be done for each single data service or artifact, such that the different activities can occur in mixed order.

Then we show an MDE setting that can be used to create dummy services, so that the UI can be developed and executed as prototype without the need that the desired services are already implemented.

Further, this report presents an MDE setting that allows to generate a model on the basis of

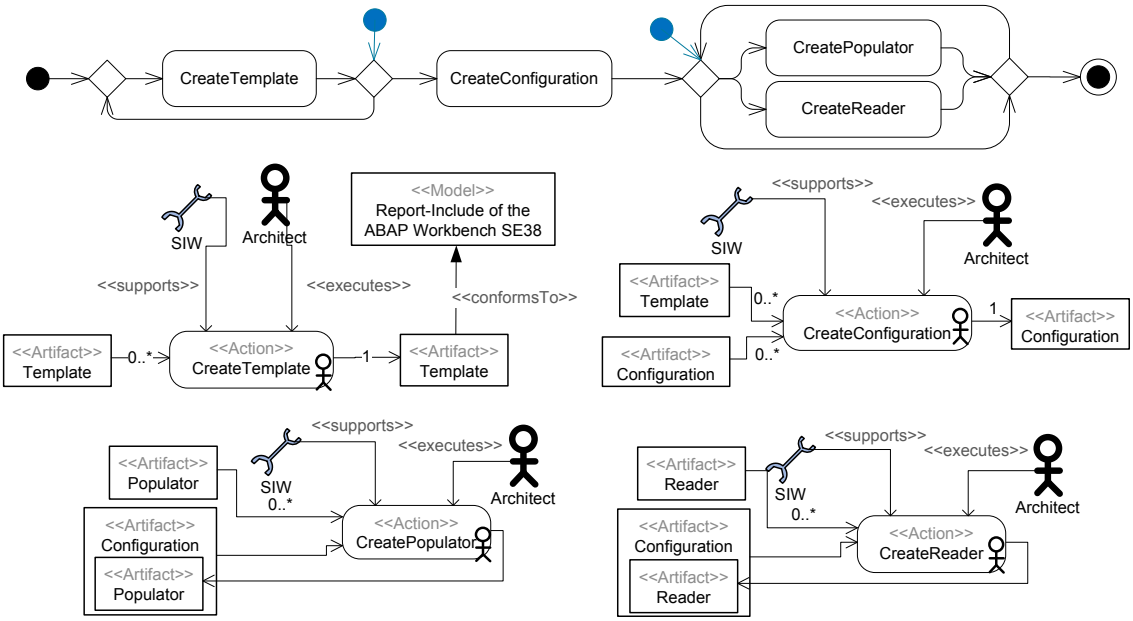

Figure 26: SIW: MDE setting of building a configuration

a BPM model. The resulting model can already be deployed to get a running application, but might alternatively be changed further by introduction of new data services or UI elements or by adaption of the layouts.

A.3.1 Creation of a model

The development of an application with VC starts with the creation of a model. Figure 28 shows an activity diagram for that. The activities are shown in more detail in Figure 29. Thereby, the first step create development component is optional. It is a semiautomated activity to create a development component within a software component. It is not necessary to create a new development component for each model, since multiple models might be part of the same development component.

The second activity is create model, which is semiautomated, too, and like the former activity supported by the VC. For the creation of a model, a repository, a software component and a development component are necessary. The resulting VC model is then created into the development component that is part of the software component. As indicated by the inheritance hierarchy shown in Figure 29 a VC model might be a service component, a composite view or have another type.

A.3.2 Integration of a data service

For the integration of a data service into an existing VC model the manual activity import data service, which is shown in Figure 30, can be used. Thereby, a representation of the data service is introduced to the VC model. As indicated by the inheritance hierarchy shown in that Figure, a data service might be accessible via a BAPI interface, via RFC, might be defined as an WSDL, an SAP enterprise service, or can even be another VC model.

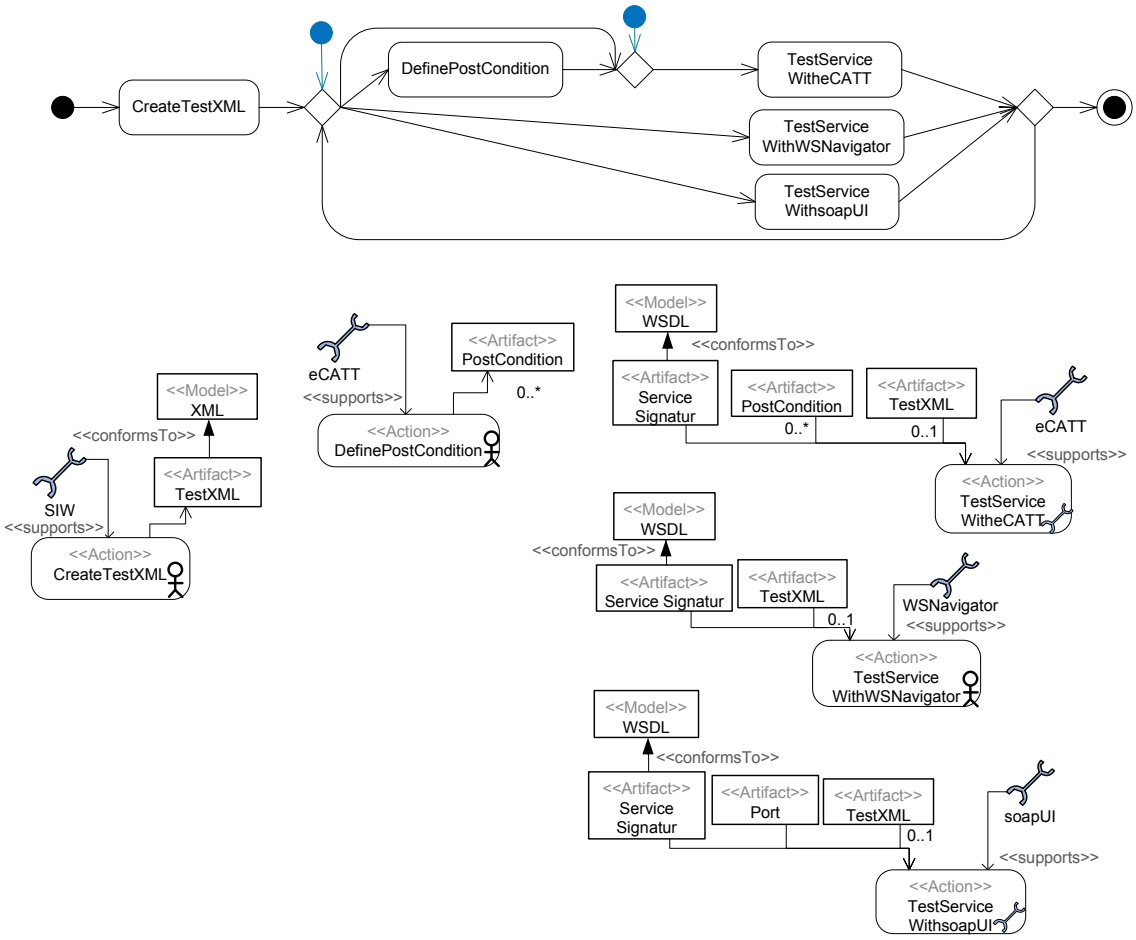

Figure 27: SIW: MDE setting for testing the created service

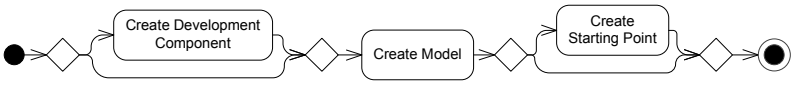

Figure 28: VC: Activity Diagram to create a model

Data services must usually be prepared to be accessible via Net Weaver and , thus, within Visual Composer. Figure 31 shows the example activity create RFC destination that is used to register an RFC. The activity is supported by Net Weaver and creates based on a name, the RFC destination, which can then be called as a data service within VC.

A.3.3 Manipulation of the data flow

Data services are not visible for the user of the application. The visible elements are here called VC model elements. They can be connected to data services in order to be filled with information or to transmit information to a data service. Figure 32 shows an activity diagram for the introduction of such a VC model element to a VC model and its connection to data services.

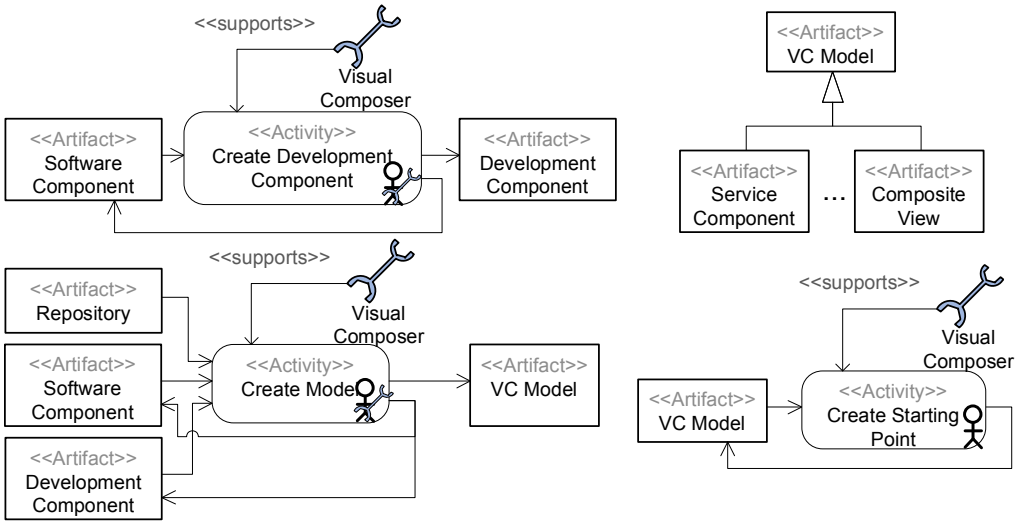

Figure 29: VC: Activities to create a model

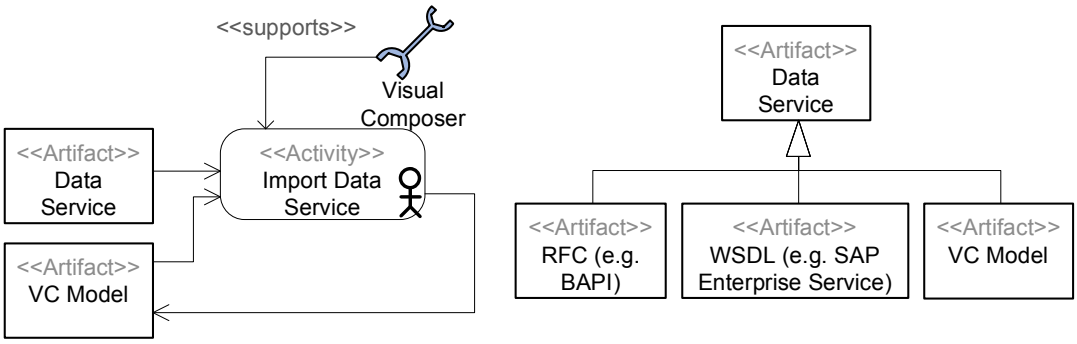

Figure 30: VC: Activities to integrate a data service into a VC Model

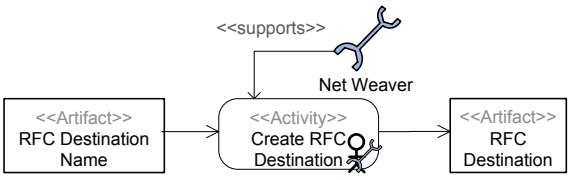

Figure 31: VC: Activity to prepare an RFC destination for its use via Net Weaver

Figure 32: VC: Activity Diagram to manipulate a data flow

Figure 33 shows the activities more in detail. The activity add element is used to manually create a VC model element within a VC model. Further, activity define connection to data service is used

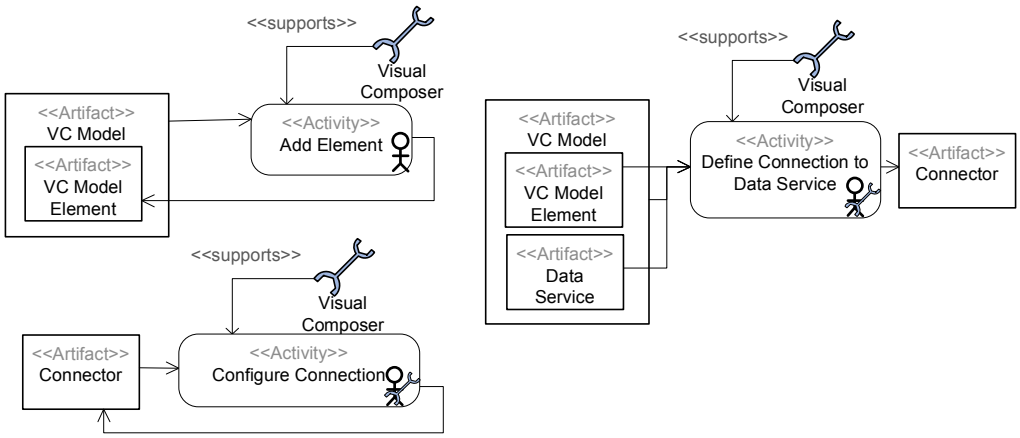

Figure 33: VC: Activities to manipulate a data flow

to define a connection between a VC model element and a data service. Thereby, a connection is create within the VC model. The activity configure connection is then used to configure the events that trigger the data flow via the connection, the mapping between the data from the data service and the VC model element, and further connection properties.

A.3.4 Manipulation of the layout

Besides the data flow to the data services, the layout of the VC model elements has to be defined. The activity diagram for that is shown in Figure 34. Figure 35 shows the activities more in detail.

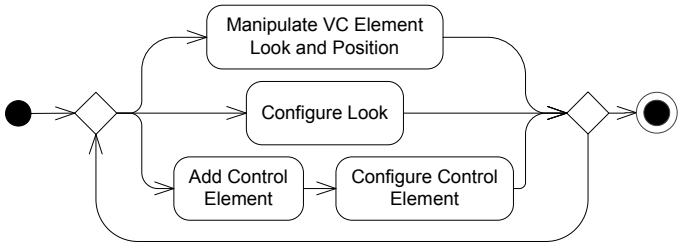

Figure 34: VC: Activity diagram to manipulate the layout

The activity manipulate VC element position can be used to manipulate manually the position of a VC model element and, thus, changes the VC model. Also the look of a VC model element can be further manipulated (activity configure look). In addition, control elements can be introduced (activity add control element) and it can be configured (activity configure control element) when the control element can be used and what happens.

A.3.5 Remove from model

Figure 36 shows the activity remove from model, which can be used to remove elements from the VC model.

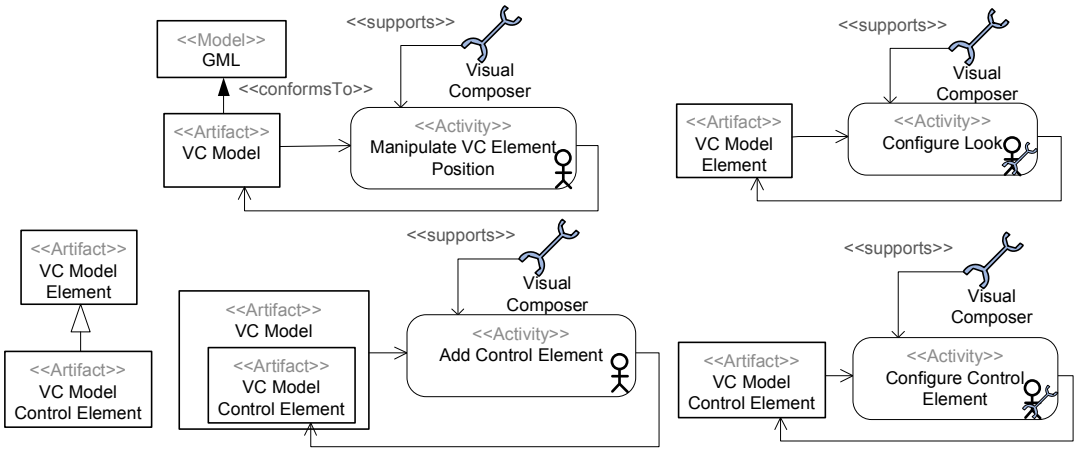

Figure 35: VC: Activities to manipulate the layout

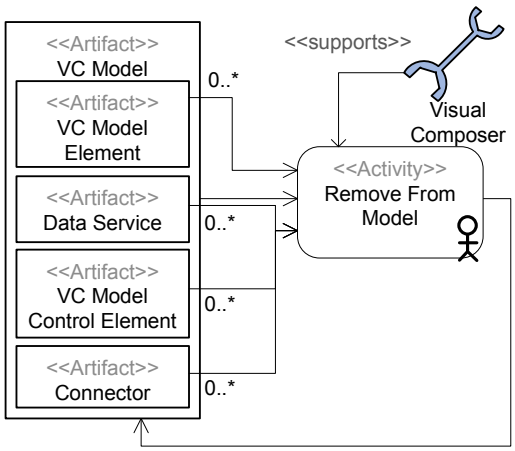

Figure 36: VC: Activity to remove parts of a VC Model

A.3.6 Deployment

After the layout and data flow is defined it is already possible to deploy the application. Figure 37 shows the two activities necessary for that. First, a runtime environment as for example WebDynpro has to be chosen (activity select runtime environment). Then activity deploy semi-automatically takes all VC models of one deployment unit and compiles them into the runtime environment.

A.3.7 Integration of web dynpro components

Alternatively to a data service a web dynpro component might be integrated in order to reach more flexibility for the application. Figure 38 shows an activity diagram of activities necessary for that.

The activities are shown in more detail in Figure 39. First, the web dynpro component has to

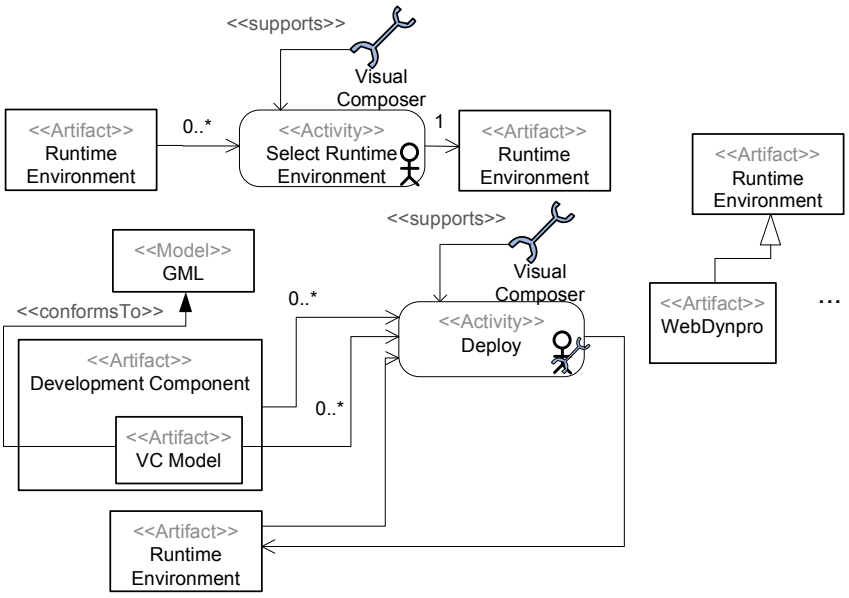

Figure 37: VC: Activities to deploy a VC model

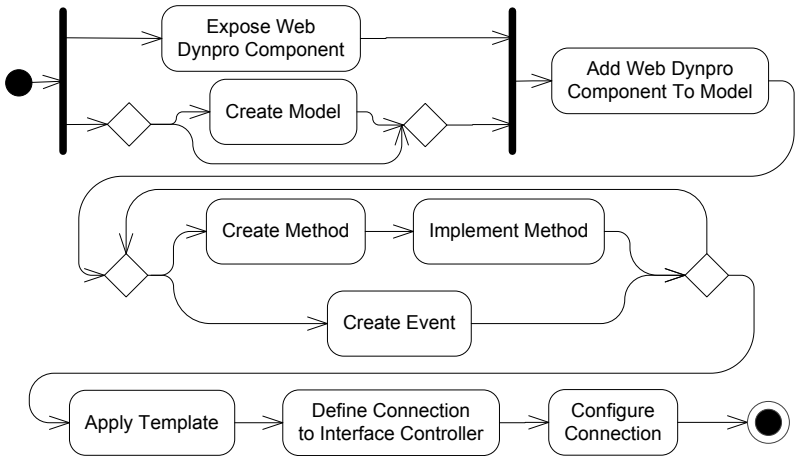

Figure 38: VC: Activity diagram to integrate a web dynpro component into a VC model

be made public via th activity expose web dynpro component. This happens within the tool web dynpro. In addition, a composite view is required which can be created using the activity create model that is shown above in Figure 29. The public component can nor be added to the composite view using the semiautomated activity add web dynpro component to model. Thereby, a component controller is added to the composite view, which represents the public component. As indicated by the show inheritance hierarchy, a component controller can be handled as data service.

In order to access functionality or data from the public component the component controller has to be extended with methods and events. The activity create method can be used to semi-automatically create a method within the component controller. Accordingly, an empty method implementation is added to the public component. Similarly, an event can be created using create event. This leads to the addition of an event in the component controller and an empty event

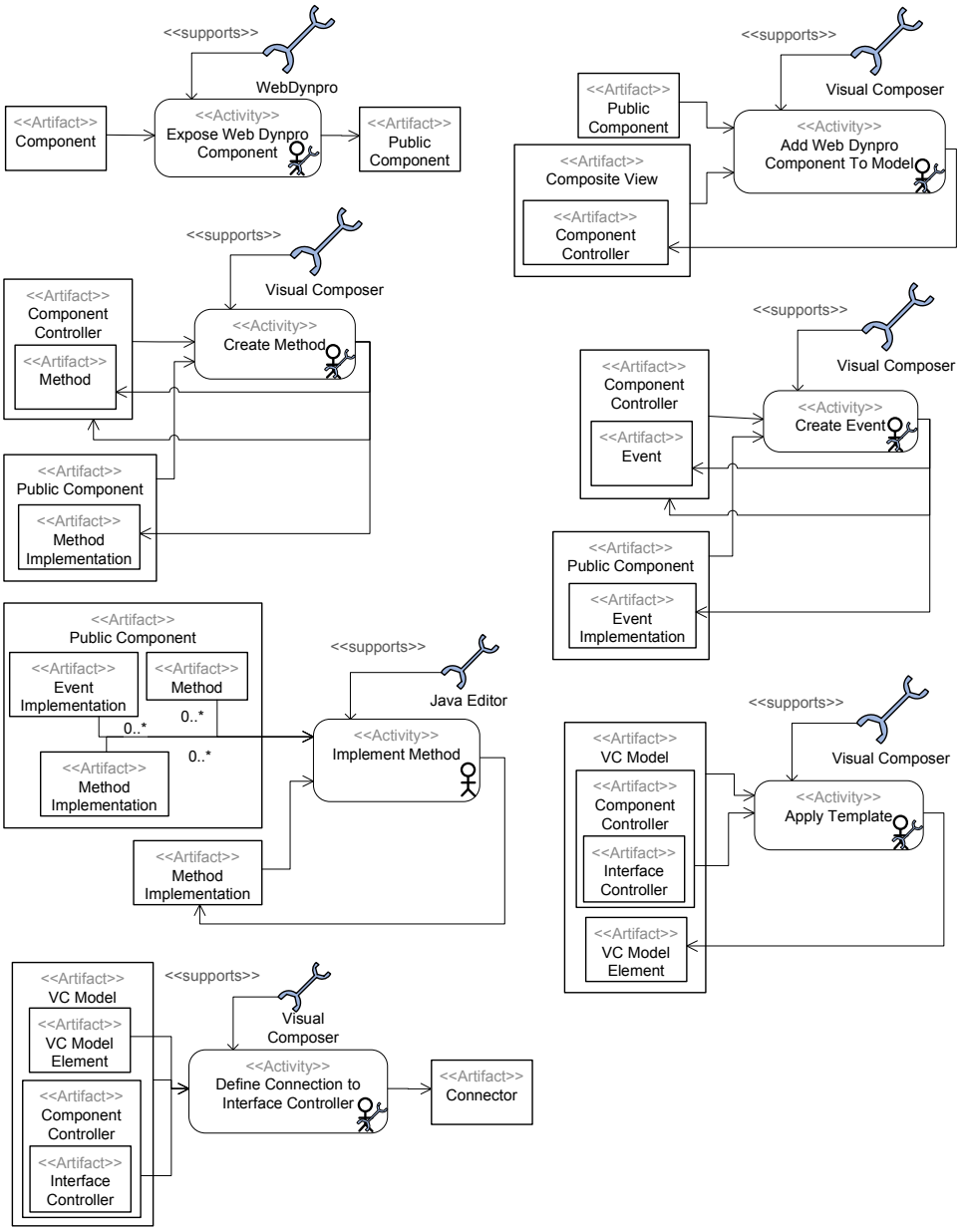

Figure 39: VC: Activities to integrate a web dynpro component into a VC model

implementation to the public component.

In order to implement the method and, thus, to access the functionality of the public component, the user has to implement the method in Java manually (activity implement method). Thereby, the method implementation is manipulated, such that it accesses other method implementations, original methods of the public component, or event implementations. Finally, the activity apply template can be used to create an appropriate VC model element for the component controller. The activities define connection to data service and configure connection can then be used to connect this VC model element to the component controller, as already described above.

A.3.8 Usage of a dummy service

It is also possible to create a dummy service and use it instead of real data service, e.g. to create a prototype of the user interface. Figure 40 shows the activity diagram for this. The activities are shown in more detail in Figure 41. First, the user might prepare dummy data in excel. In addition, if necessary a composite view has to be created using the activity create model that is shown above in Figure 29.

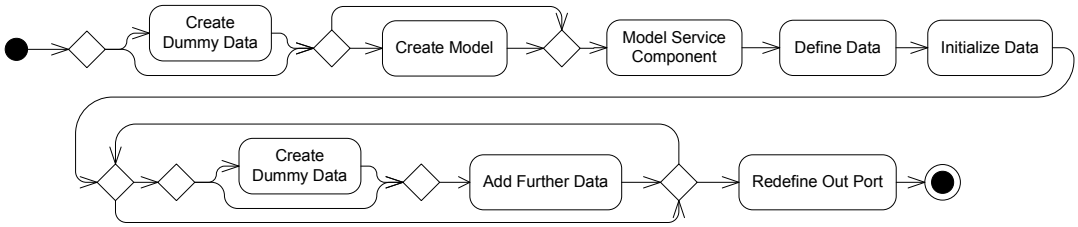

Figure 40: VC: Activity diagram to create a dummy service

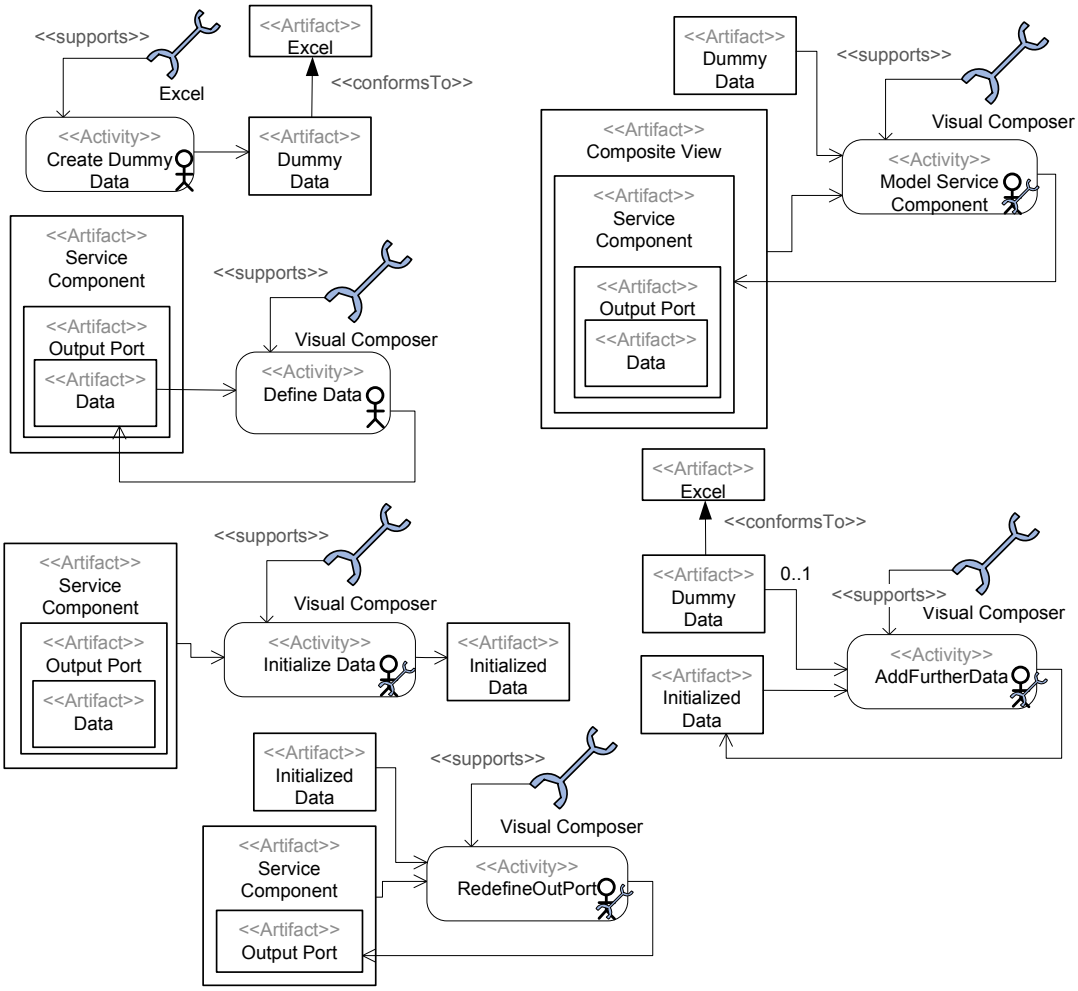

Figure 41: VC: Activities to create a dummy service

Then a service component can be created semi-automatically within the composite view. Thereby, the prepared dummy data can be copied, such that the data for the output port is defined (activity model service component). In addition, input ports can thereby be created. Then the user can refine the data structure of the out port (activity define data). Then the data of the service component has to be initialized, using the semi-automated activity initialize data. Thereby, the dummy data used for the creation of the service component is added automatically. Data for the refined parts of the data structure can be added using the activity add further data. As shown, further dummy data can be created with create dummy data to give additional input for add further data.

Due to technical constraints, the output port finally has to be redefined, such that it provides the initialized data. Now the service component can be handled in the data flow description as a data service, as described above.

A.3.9 Generation of simple UI on the basis of BPM model

The last MDE setting part that we handle in this report enables the generation of a simple VC model on the basis of a BPM process model. Thereby, as shown in Figure 42 three activities are necessary. Figure 43 shows these activities in more detail. First the process has to be modeled (activity model process). Following, it is necessary to generate a process context, which includes the references on the data services that have to be used. This step is automatic.

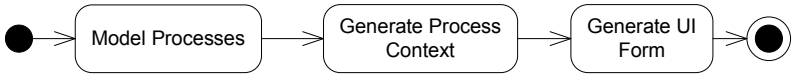

Figure 42: VC: Activity diagram to generate a VC Model on the basis of a BPM model

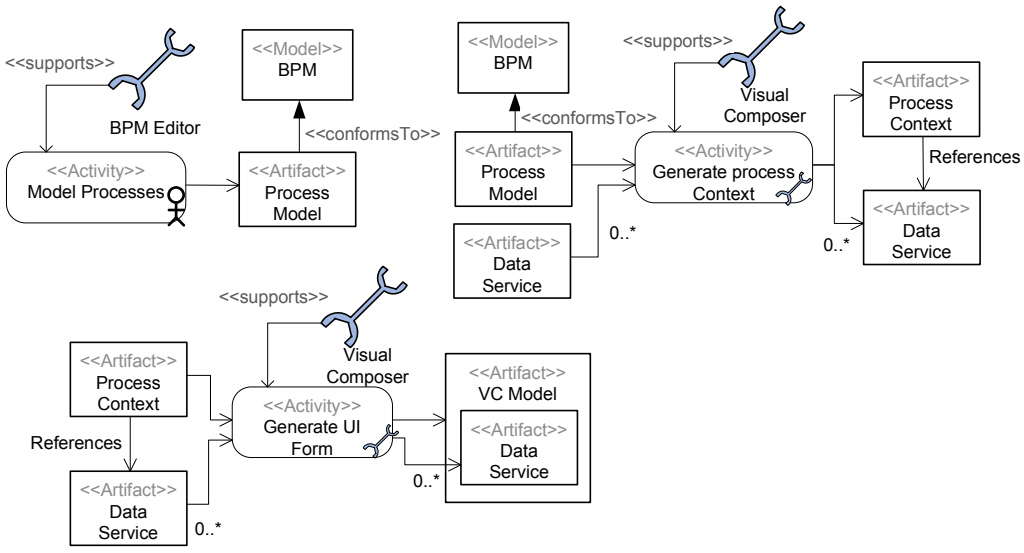

Figure 43: VC: Activities to generate a VC Model on the basis of a BPM model

Finally, the automatic activity generate UI form takes this context and generates the VC model. This generated model can already be deployed to a running application. However, it is also possible to apply further changes using the activities described in the sections above.

A.4 Oberon

The Oberon UI-Designer is a tool used to create user interfaces for SAP ByDesign. Thereby, the UI-Designer provides templates (called floorplans) for user interfaces (called UI-design here). The UI-designs are models of the user interface, which can be manipulated and enriched with a data model and behavioral aspects, like navigation between different UI-designs or access to business objects. Finally, the resulting model can be interpreted by tools like Silverlight or the Oberon Slim Client.

Through the continuous usage of the model during development UI-Designer allows the developer to act on a high level of abstraction. Even the resulting model is directly interpreted and the developer has not to deal with generated code that might have to be adapted further. Thus, the UI-Designer is an example for a sophisticated MDE that requires the developer only little to access additional artifacts besides the main model.

First parts of the UI-Designer MDE setting can already used during analysis and design phase, when the appearance and the usage scenarios of the user interface is planned. Here UI-Designer can be used to create models for the appearance design. These models can then directly be reused during implementation phase, where they are further refined and enriched with access to back end functionality via business objects. Finally, it is not only at implementation time, but also after deployment time possible to define changes in the user interface for specific users.

We can describe the MDE setting for developing a user interface with Oberon UI-Designer in three parts. First part concerns the design of the user interface. The activities can be performed during design but also later at implementation phase. The second part concerns the implementation of the user interface, e.g. a refinement of the user interface, as well as the assignment of behavior, such as navigation or access to functionality of the system. The last part describes how the user interface can be changed - even by a key user - for specific users and how the user interface is executed. An activity diagram of the whole MDE setting is shown in Figure 44.

A.4.1 Designing the user interface

Figure 45 shows the activities that might already been applied during design. The implementation or design starts with the choice of an appropriate template for a floorplan. Based on that the floorplan is automatically created, which can now be manipulated to define the look of the user interface. As a floorplan represents only one part of the user interface it has to be integrated into a whole user interface, which is a work-center with work-center views. Thus, the activity 'AddUIDesignToWorkCenterView' adds the floorplan to a work-center view semi-automatically.

It is possible and in most cases necessary to create more than one floorplan. During design we can further define how to navigate between different floorplans. Therefore, it is possible to manually define hard coded navigations directly between the floorplans. This activity is called 'DefineNavigation'. It is one of a couple of activities that can be performed after the floorplan and the associated data model are created. However, 'DefineNavigation' can not only be applied to floorplans, which are a special kind of UI components, but to all kind of UIComponents.

A.4.2 Implementation of the user interface

The other activities that can now be executed to implement UIDesigns are shown in Figures 46 and 47. First of all, the floorplan can be further manipulated (activity 'ManipulateFloorplan') by

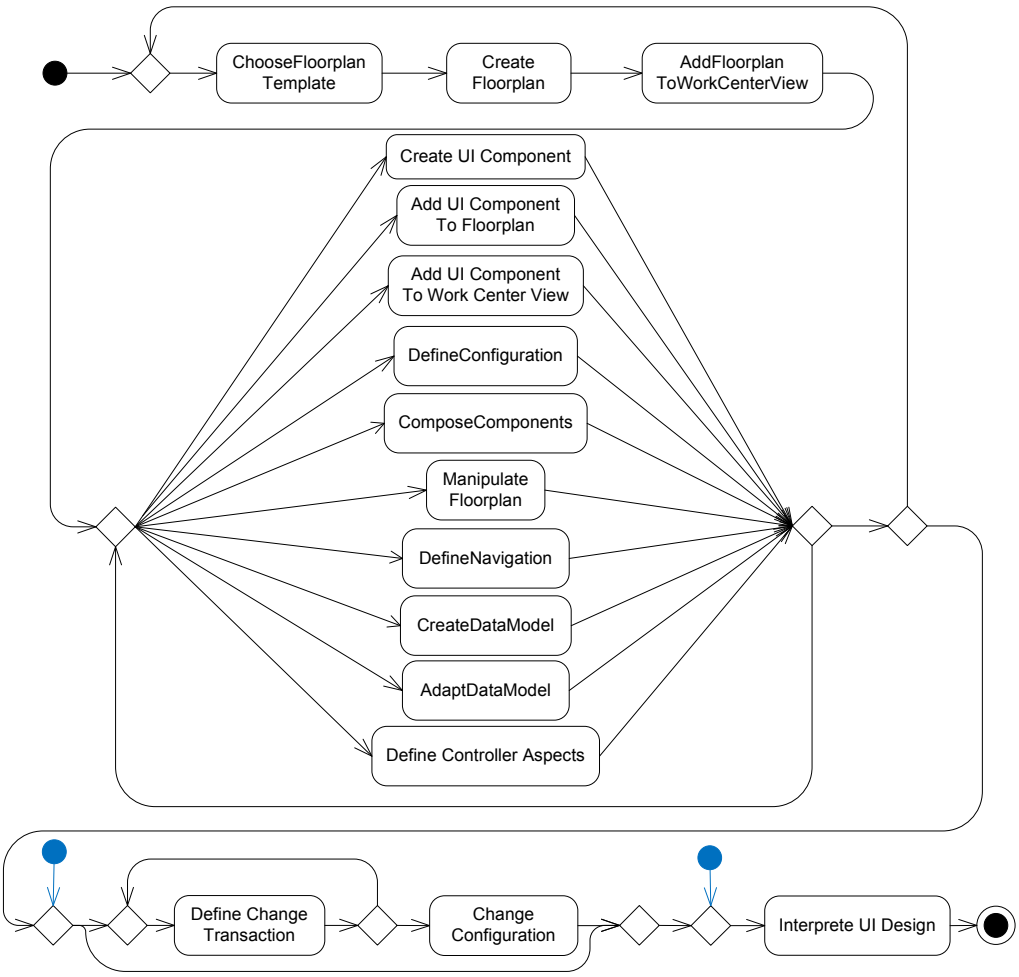

Figure 44: Oberon: Order of activities in the MDE setting for building a user interface using Oberon

the developer.

Further, additional UIComponents can be created semi-automatically (activity 'CreateUIComponent') to refine the structure of the floorplan. The components can than be added directly to the floorplan (activity 'AddUIComponentToFloorplan'), to a work center view (activity 'AddUIComponentToWorkCenterView') , or to other UIComponents (activity 'Compose Components'). The composition between UIComponents can be loose via in and out ports or tight via bindings on elements of the UIComponents data model. As a floorplan is a UIComponent as well, the activity 'AddUIComponentToUIDesign' is similar to the activity 'ComposeComponents'.

The activity 'CreateDataModel' can be used to semi-automatically create a data model for the floorplan or each other UI component. Such a data model can be further adapted (activity 'AdaptDataModel').

Further, the activity 'DefineControllerAspects' capsules multiple different activities that are necessary to define the behavior of the user interface. Figure 47 shows these refining activities. Two of the activities are 'CreateQuery' and 'BindToQuery', which can be used to create a query and to bind it semi-automatically to a specific part of the UIComponent. Further, the semi-automated ac-

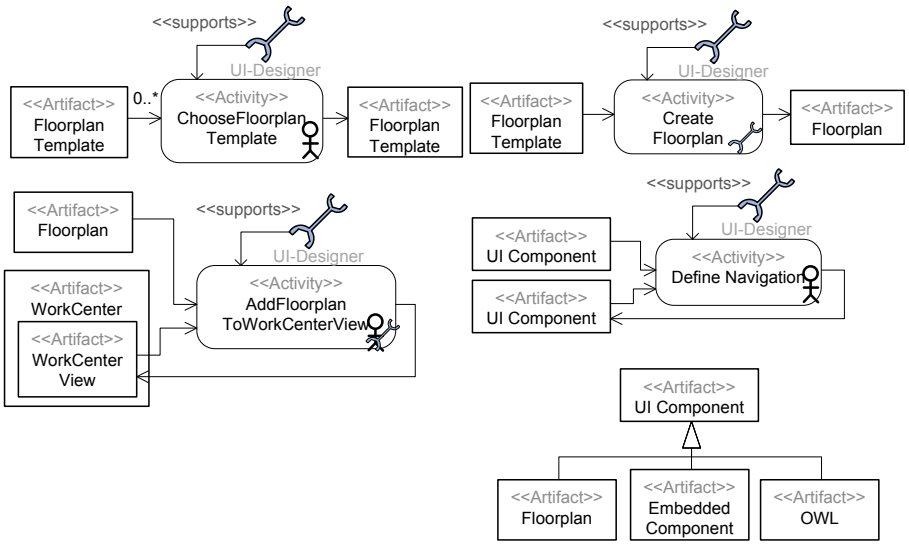

Figure 45: Oberon: Activities of the MDE setting for designing a user interface using Oberon

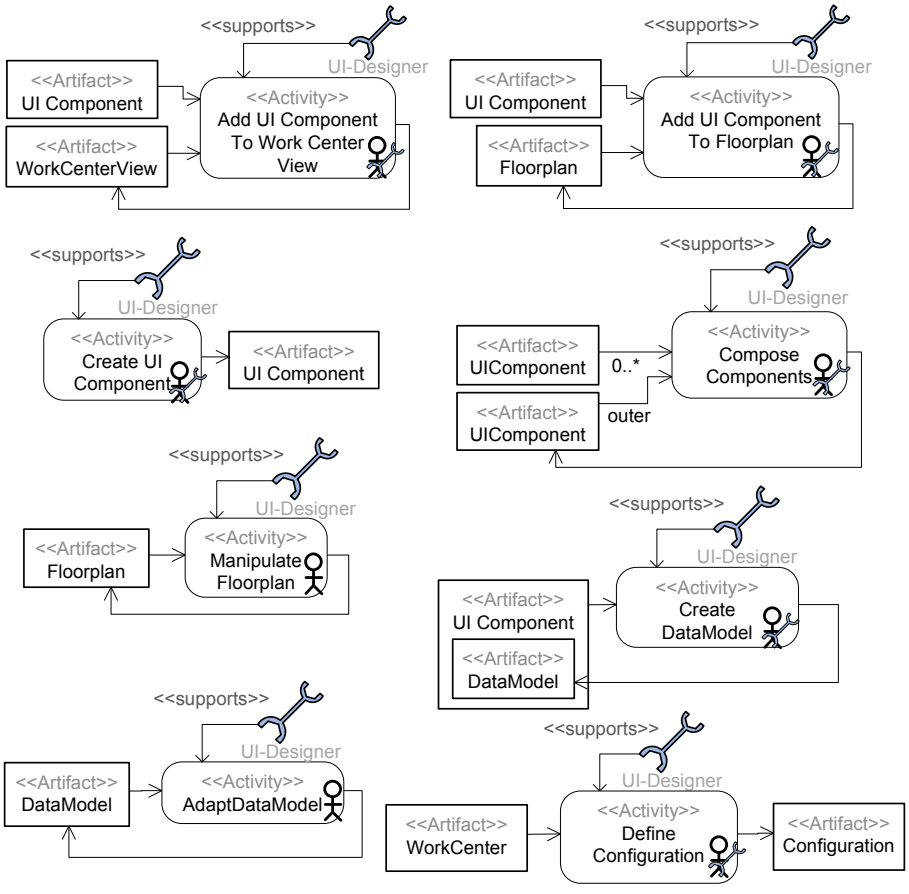

Figure 46: Oberon: Some activities of the MDE setting for implementing a user interface using Oberon

tivity 'BindToBusinessObject' is used to access a business object from the user interface. Therefore, the corresponding data model is bound to the business object (based on the BO-MetaData) and the UIComponent references the data model, accordingly.

In addition, the semi-automated activities 'CreateEventHandler' and 'AddEventHandler' can be used to couple the UIComponent to event handlers. Further, it is necessary to create a configuration file (activity 'Define Configuration').

The last four actions are for definition of OBN-based navigation between UIComponents. Therefore, 'AddOBNReferenceToModel' adds references to the source UICOmponent, while 'DefineAsOBNTarget' defines within a UIComponent that it is a target of a specific OBN navigation. Finally, 'RegisterOBNReference' and 'RegisterOBNTarget' are automatically executed and store the information about source and target references within a NavigationRegistry.

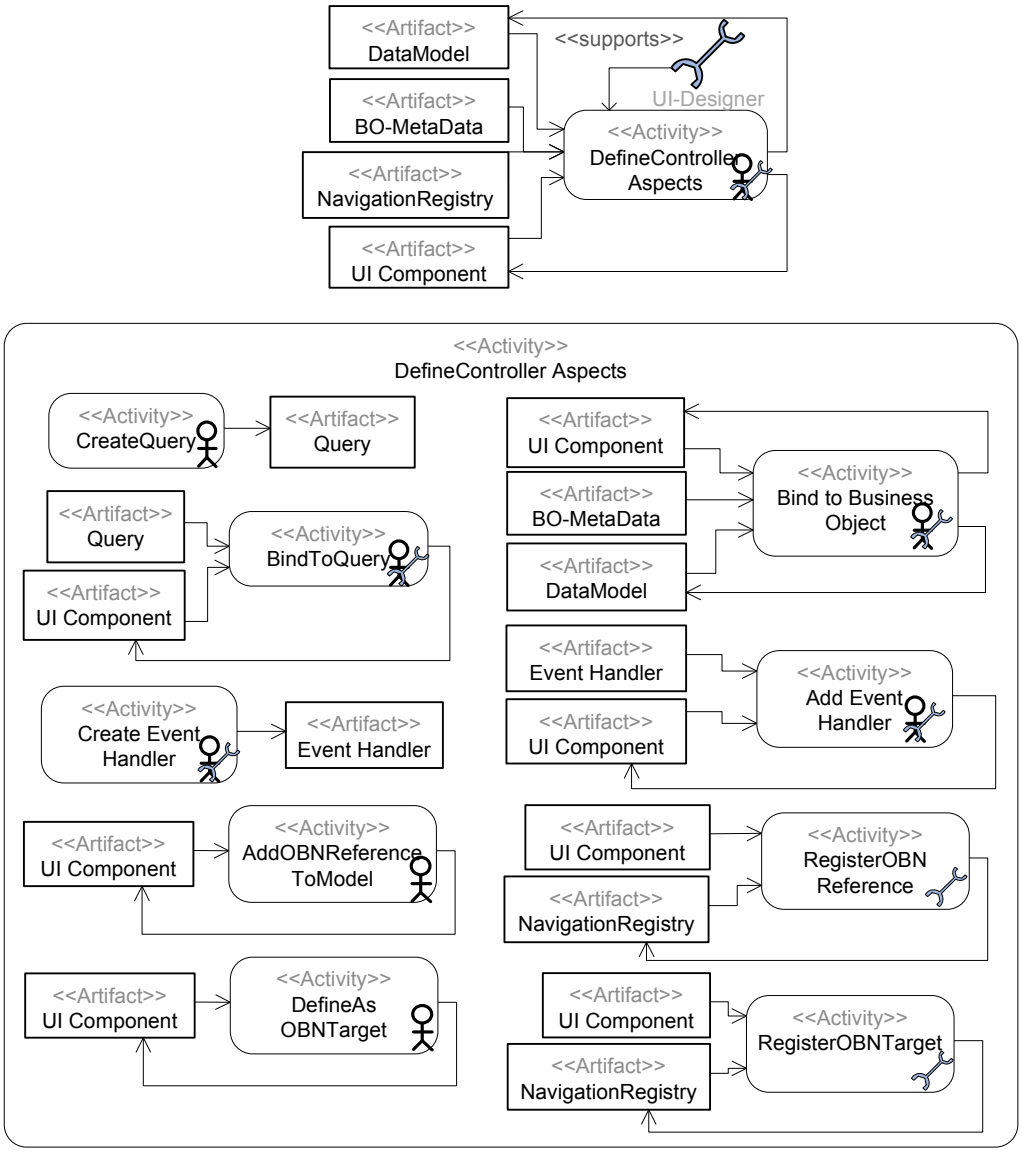

Figure 47: Oberon: Some activities of the MDE setting for implementing a user interface using Oberon

A.4.3 Defining changes for specific users

After the user interface and its functionality is defined it can be interpreted directly using Silverlight or other tools like the Oberon Slim Client (activity 'InterpreteUIDesign' shown in Figure 48). In addition, it is possible to semi-autoamtically define so called ChangeTranstions (activity 'DefineChangetransition') on the floorplans for specific users. This can be done before deployment by the user interface designer, but also after deployment by the the key user of the system. For which user a ChangeTransition is defined can be configured (activity ChangeConfiguration).

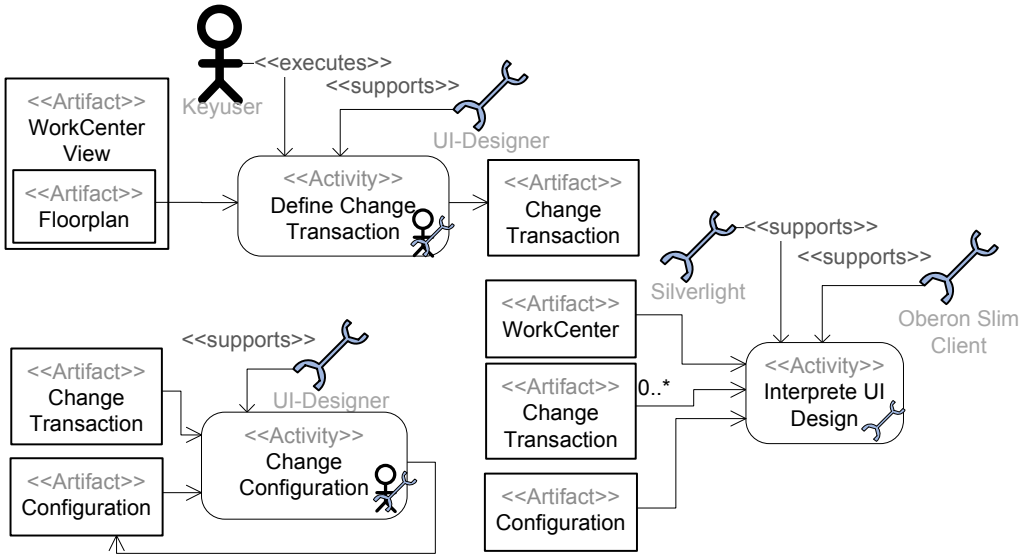

Figure 48: Oberon: Activities of the MDE setting for interpreting and changing a user interface using Oberon

A.5 Business Rule Framework (BRF)

The Business Rule Framework enables customers of SAP to easily define changes in their business processes by defining new rules or changing older rules. Thereby, BRF+ can directly be used by the customer (the user of the business process) for changing the business process.

There are two basic MDE settings for the introduction of rules using BRF+. First, the business user (i.e. the customer of SAP) can introduce the new rules using BRF+. Alternatively the rules might be introduces by the SAP process developers. In the first case a preparation of the process is necessary to enable the later introduction of rule by the user. For SAP processes the process developers might introduce business rule services into the process. This might alternatively be done by the business user with the help of BRF+. There is a third MDE setting for the preparation of non-SAP processes by the process developer.

BRF+ automatically converts the rule descriptions (which can be seen as models) into database entries, which are later generated to code in form of rule classes. Thus, we can consider the form to define rules in BRF+ as a domain specific language (DSL) and with it the MDE settings that are supported by BRF+ as MDE approaches.

This section includes the different MDE settings. The Sections A.5.1, A.5.2, and A.5.3 describe the MDE settings for the preparation of the process. These MDE settings might be applied during implementation phase or deployment phase. Parts of the MDE setting for the preparation of the process by the business user might also be applied at runtime.

The MDE setting for the manipulation of the rules is shown in Section A.5.4. It can be applied at runtime of the process. Finally, Section A.5.5 shows the MDE setting for the initial introduction of rules by the process developer. This MDE setting might be applied during deployment phase of the process.

A.5.1 The preparation of the process by the process developer

Figure 49 shows the MDE setting of the preparation of a SAP process. As result of the preparation at different points in the process special business rule services are requested. These services later access the rules. First, the SAP process has to be implemented. As this activity might be complex and is not in focus of this report, it is not annotated here which tools are used for this step or if this step is done manually or with automated support.

It is further not captured in which language (e.g. BPML) the process is formulated, as the developers might even decide to implement the process directly without using a process description language.

After the process is implemented, the process developers create business rule services that are accessed by the process.

A.5.2 The preparation of the process by the business user

Figure 50 shows the MDE setting for the preparation of an SAP-process by the business user. This MDE setting differs from the one presented above, since the business user creates and manipulates the business rule services. BRF+ supports these activities. They have not to be performed manually, but are semi-automated.

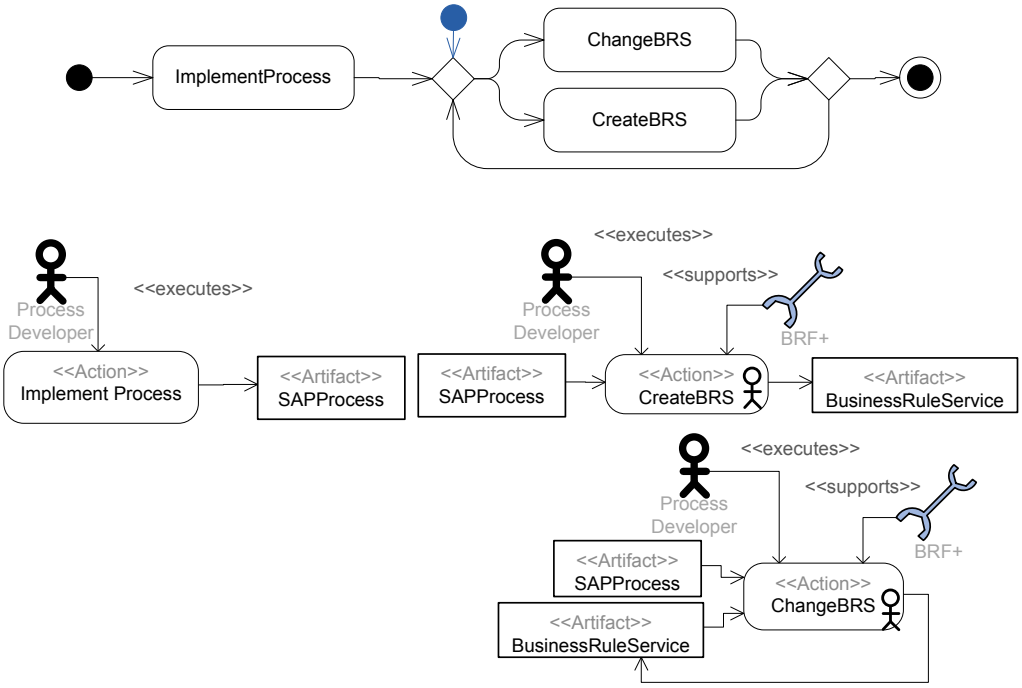

Figure 49: BRF: MDE setting of the preparation of an SAP process by the process developers.

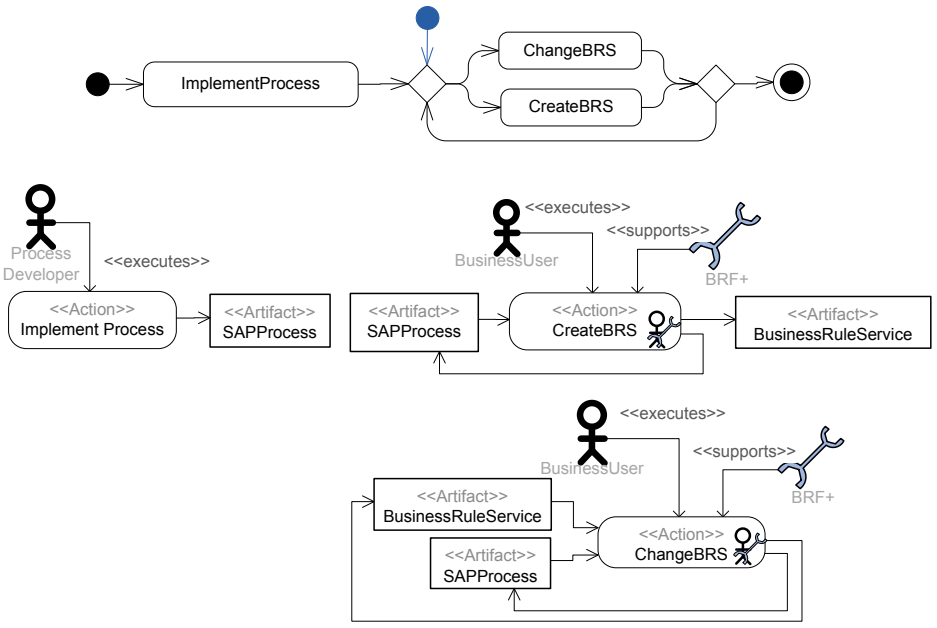

Figure 50: BRF: MDE setting of the preparation of an SAP process by the business users.

A.5.3 The preparation of a non-SAP process by the process developer

Figure 51 shows the third preparation MDE setting, which describes how an existing non-SAP process can be prepared for the usage with BRF+. Therefore, the process developer has to create a business rules service and then he has to tie this service to the code of its process. This can be applied multiple times.

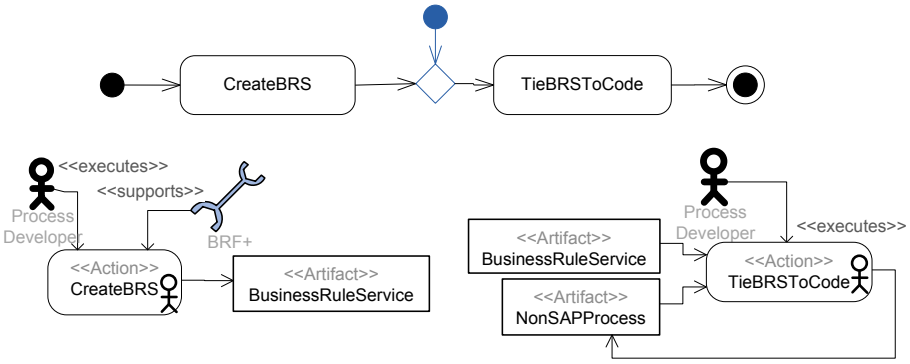

Figure 51: BRF: MDE setting of the preparation of an non-SAP process by the process developers.

A.5.4 The manipulation of rules

Figure 52 shows the order of activities of the MDE setting for the manipulation of rule by the business user. The corresponding activities of the MDE setting are shown in Figures 53 and 54.

First, the business user has to create a rule set for the business rule service. This rule set can then be enriched with decision tables and rules, which might also be changed later on.

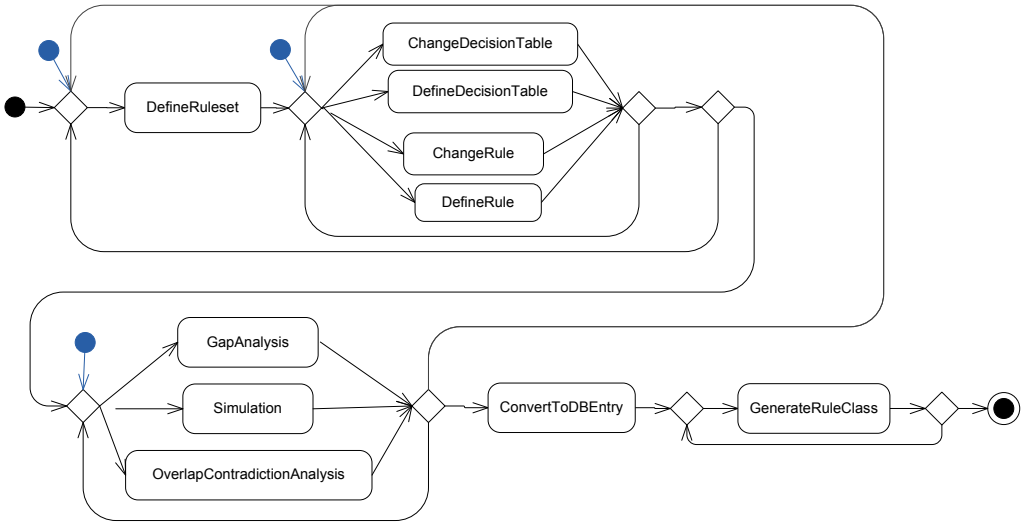

Figure 52: BRF: Order of activities in the MDE setting for the manipulation of rules

If the rules and decision tables are filled, BRF+ supports the user in applying several kinds of

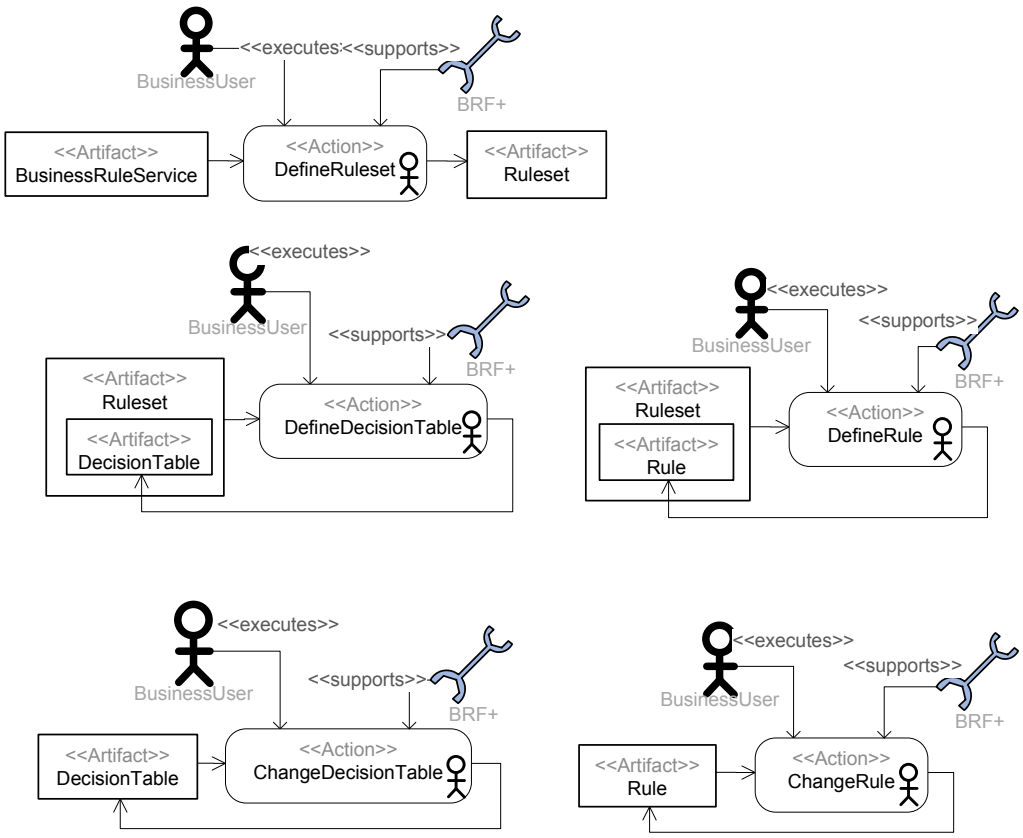

Figure 53: BRF: Activities of the MDE setting for the manipulation of rules

analysis activities, such as a gap analysis or an overlap contradiction analysis. The user is also supported in simulating the decisions based on the rules. All three activities are semi-automated. If the analysis or simulation shows mistakes in the rules, the business user might go back and change the rules and decision tables.

Finally, the business user triggers BRF+ to convert the rule set to a data base entry. When the business rule service is later on called the first time by the process, BRF+ performs the last step of the MDE setting and generates a rule class which is then accessed by the business rule service. Both, conversion to the data base and generation of the rule class are fully automated steps that do not require the user to be active.

The business user might always decide to return to the point where he defined the rule set or rules and decision tables. However, as the activity generate rule class is not directly influenced by the business user, there is no additional entry point, that enable to start at this point of the MDE setting.

A.5.5 The initial introduction of rules by the process developer

Figure 55 shows the order of activities of the MDE setting for the initial introduction of rules by the process developer. The Figures 56 and 57 show the activities that this MDE setting.

This MDE setting starts with a business user that defines (and might later change) unstructured

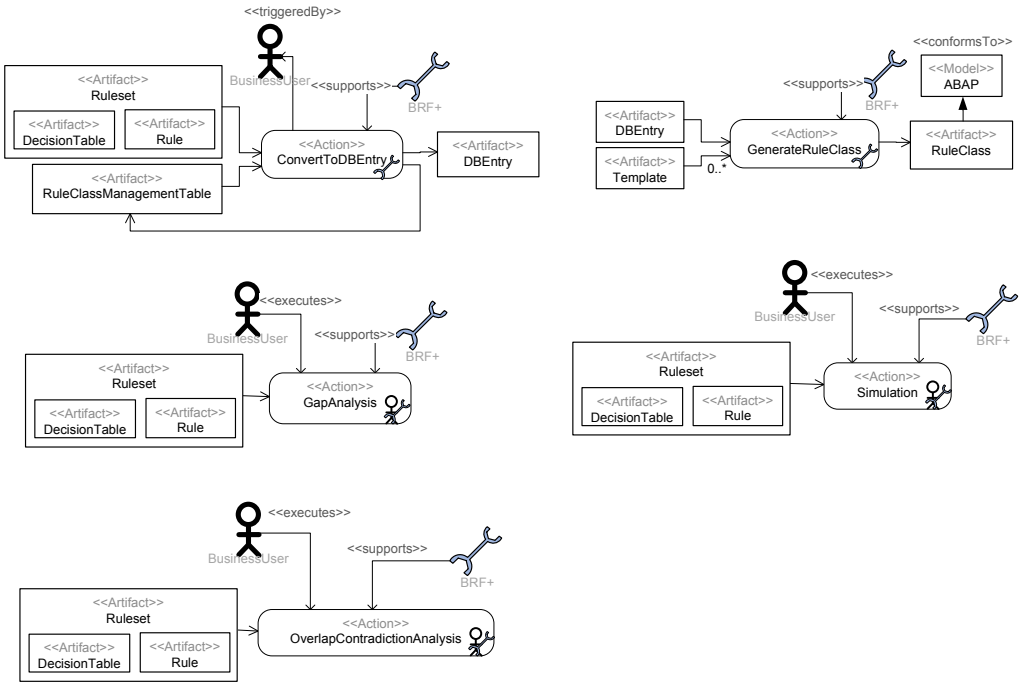

Figure 54: BRF: Activities of the MDE setting for the manipulation of rules

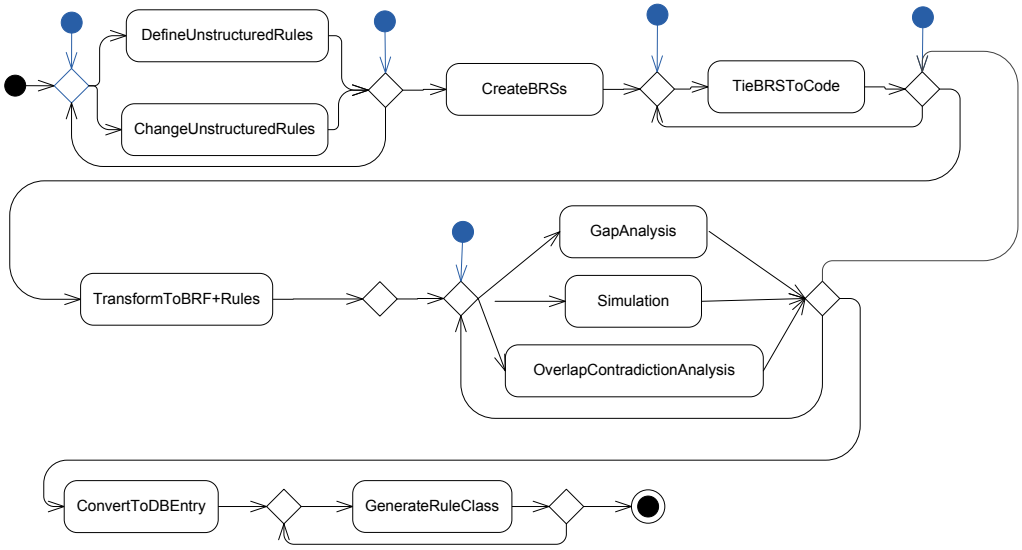

Figure 55: BRF: Order of activities in the MDE setting for the initial introduction of rules

rules that should be integrated to the process. For this rules the process developer defines then business rule services, which he ties to the process code. After the business rule service is created the process developer can transform the unstructured rules given by the business user to BRF+ rule sets with rules and decision tables.

The following activities shown in Figure 57 are similar to the last activities of the MDE settings shown in Section A.5.4. After the rules are transformed they might be analyzed or simulated with

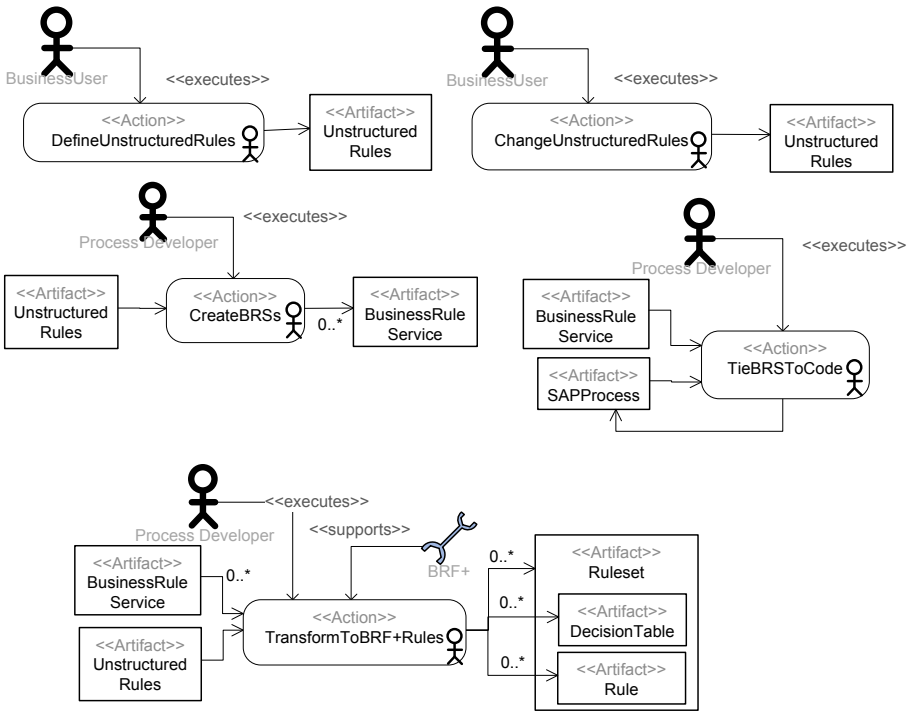

Figure 56: BRF: Activities of the MDE setting for the initial introduction of rules

the possible consequence of changes in the rules and decision tables. Finally, they are converted to data base entries and later these entries are the bases for the generation of rule classes. The difference is that these activities are executed by the process developer and not by the business user.

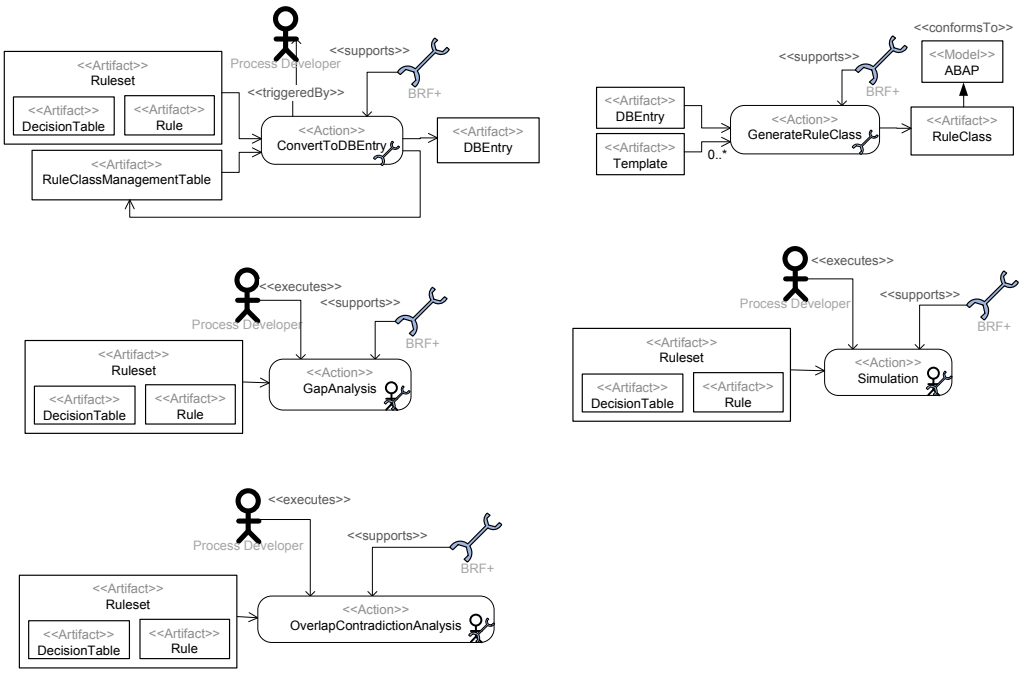

Figure 57: BRF: Activities of the MDE setting for the initial introduction of rules

A.6 Business Warehouse (BW)

Business warehouse (BW) is a tool that can be used by a company, to bring the data necessary for the reporting from different ERP systems to a central data warehouse. Therefore, BW enable the user to define the structure for the data in the central data warehouse. Further, it can be defined how the data structure from a source ERP system has to be transformed to the data structure in the central data warehouse. Finally, the queries on this data for the reporting and report layout can be defined. All this is then interpreted. For the interpretation some parts of code have to be generated and are automatically regenerated, the the generation sources are updates. Thus, changes can also be applied after deployment of the system.

Figure 58 shows a slightly adapted activity diagram of the MDE setting to get a centralized reporting using BW. We annotated additional initial nodes and flow final nodes in the diagram (marked in blue). We use this to indicate that the user stop the process or execute only parts of the process.

For example, the process might be split into four parts. There are activities for the definition of the data structure in the central data warehouse. Further, there are activities for the definition of transformation between the information of a specified ERP system and the defined central data structure. Then, there are activities to define how the reporting is applied on the data in the central data structure. Finally, there are the automated activities for the execution of the reporting. The initial node before the activity create info source indicates for example that the user can define the translation for a new ERP system. If he does not want to change the way of reporting he can stop with the process before executing the activity define container subset. The last three activities are necessary for execution of the reporting and might therefore be executed again and again although the other activities are not.

Figure 59 shows the activities for the definition of the data structure for the central data warehouse

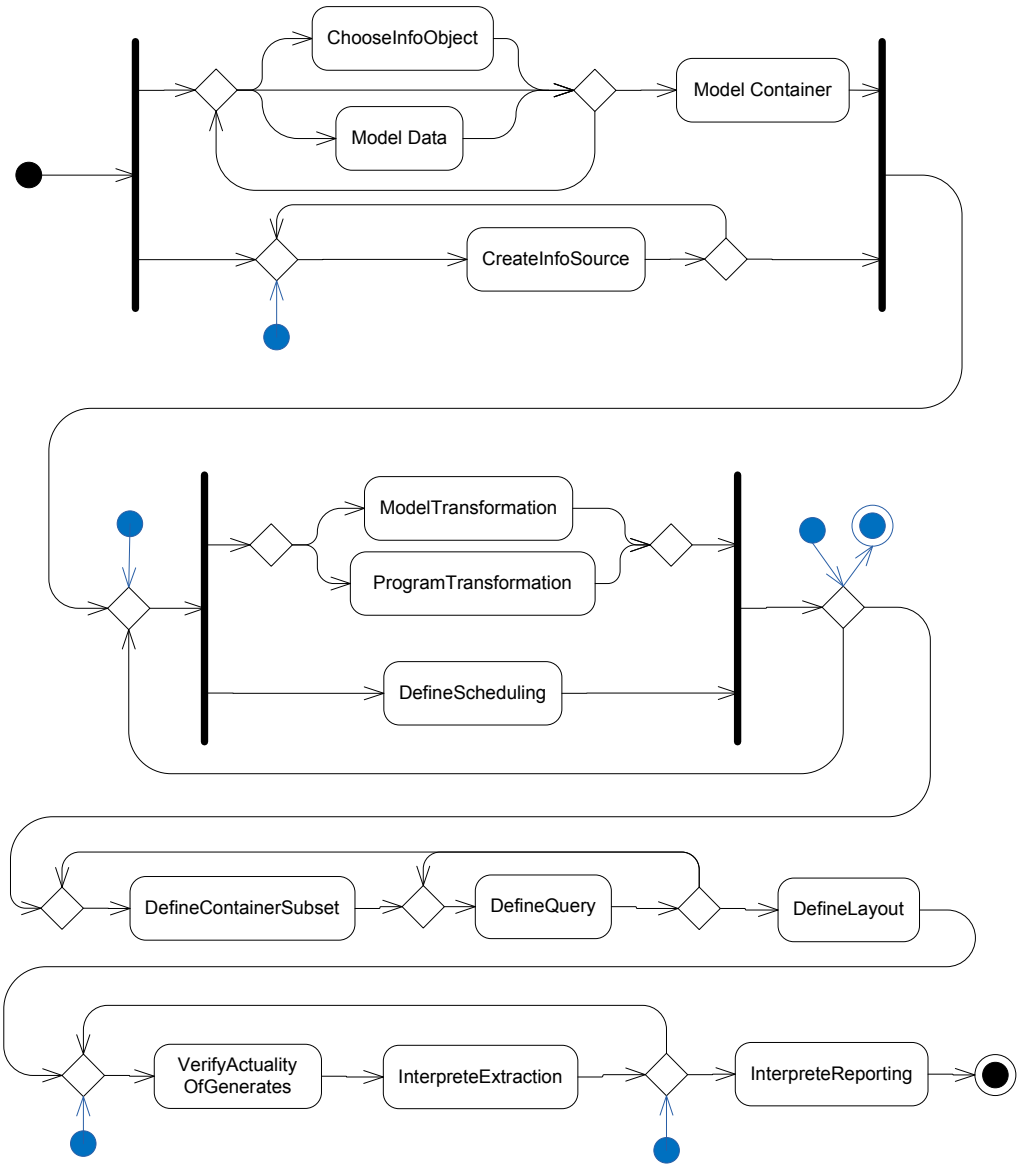

Figure 58: BW: Order of activities in MDE setting for centralization of reporting using BW

more in detail. The activity model data can be used to define a part of the information structure (info object). However, it is also possible choose from already existing parts (activity choose info object). If the required parts are created and chosen, the information structure is retrieved through combining the info objects to a container. All these activities are manually (indicated by the stick man) and are supported by the tool BW.

For each ERP System that should be integrated into the reporting, the definition of the extraction has to be done. Thereby, it is assumed that an ERP system contains a data extractor which extracts the information from the ERP system and provides them for further usage. Since multiple of the ERP system might provide the data in the same form, they can be associated to the same info source which represents the source type. An info source is created using the manual activity create info source. Thereby the info source gets references to the associated data extractors.

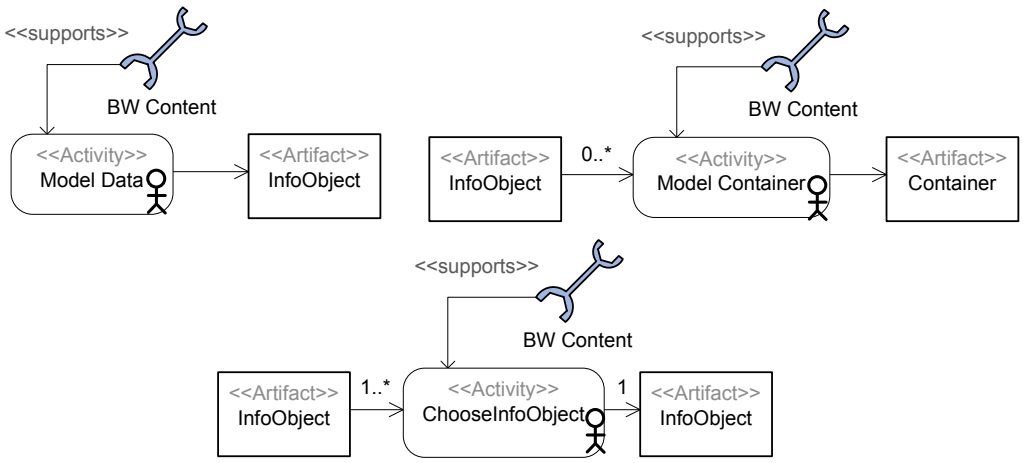

Figure 59: BW: Activities for definition of central data structure from MDE setting for centralization of reporting using BW

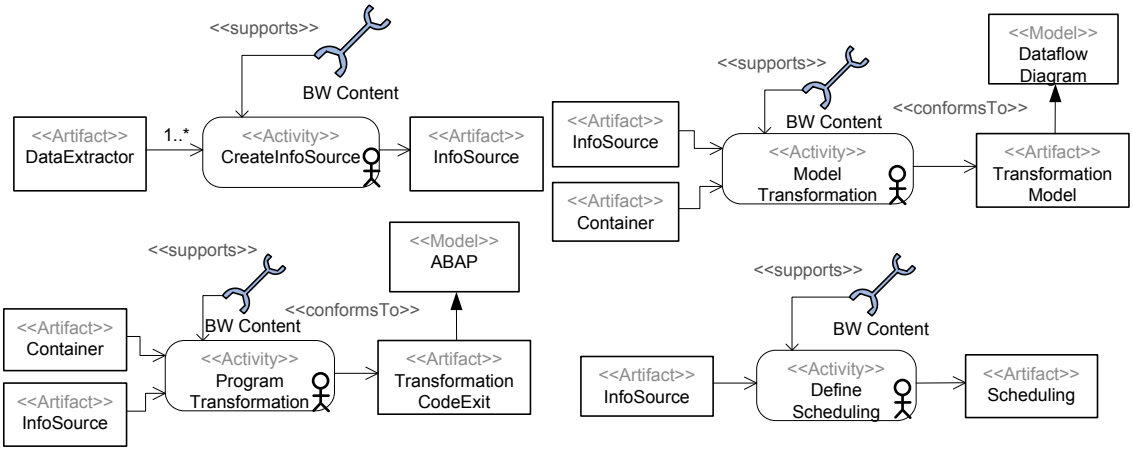

Figure 60: BW: Activities for definition of data extraction from MDE setting for centralization of reporting using BW

For each info source it is necessary to define how the information has to be transformed to the structure of information in the container (i.e. in the central data warehouse). This can be done in two ways. Using activity model transformation it is possible to model the transformation using a data-flow diagram. Alternatively, more complex transformations can be programed as a transformation code exit using activity program transformation. Besides the transformation also a scheduling has to be defined, which specifies, when the data extractors associated to the info source have to be used to extract the information (activity define scheduling). Figure 60 shows the four activities in detail.

Apart from the extraction of information, it is necessary to define how the reporting has to be applied on the data structure in the central data warehouse (container). Figure 61 shows the activities necessary for that in detail. First, the user chooses a subset of the container structure (activity define container subset). Based on this subset he can define queries (activity define query). Both activities can be applied multiple times, to create multiple queries on multiple container subsets.

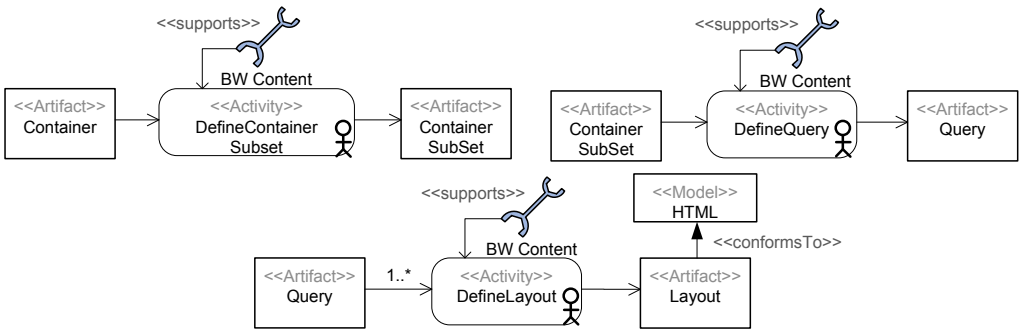

Figure 61: BW: Activities for definition of reporting from MDE setting for centralization of reporting using BW

Finally, the layout of the reporting can be defined based on the queries (activity define layout). Thereby, the layout is a HTML file.

Now all information necessary for the execution of the reporting is created. Thus, transformation models, the container, transformation code exits, info sources with their scheduling, container subsets, queries, and layout are transported to the running system.

Before the execution can be performed, it is necessary to ensure that transformation rules, transformation code, and database structures, generated out of the actual transformation models, transformation code exits and container exists. This is done by the automatic activity verify actuality of generates. It is performed before the extraction of data out of the different ERP systems and the reporting are performed. This activity can be further decomposed to the six activities shown in Figure 62.

First, the activity choose transformation models to regenerate compares the creation date of the transformation rules (if already there) and the transformation models. The the transformation rules of one transformation model are out of date or do not exists the transformation model belongs to the set of chosen transformation models. For each of these chosen transformation models the activity generate transformation is performed to create the actual transformation rules.

Similarly, the activity choose transformation code exits to regenerate, compares transformation codes and transformation code exits to retrieve the set of transformation code exits that are transformed to transformation code using the activity generate code exits.

Finally, activity decide whether to regenerate container is used to decide whether the database structure has to be regenerated. In that case, activity generate database structure generates on the basis of the container the database structure. All these activities are performed by the tool BW.

Figure 63 shows the two activities for the execution. For each info source the tool BW decides based on the associated scheduling whether to trigger the associated data extractors. This is the activity interpret extraction. Thereby, the info source with the associated scheduling as well as transformation rules or transformation code are input. Further, the database structure and the container are input. As result of this activity the container is filled with new information.

When all information is extracted, the user might decide to trigger a reporting tool (such as Excel) to apply the activity interpret reporting. Thereby, based on the container subsets, the queries, and the layout are used to create an HTML report.

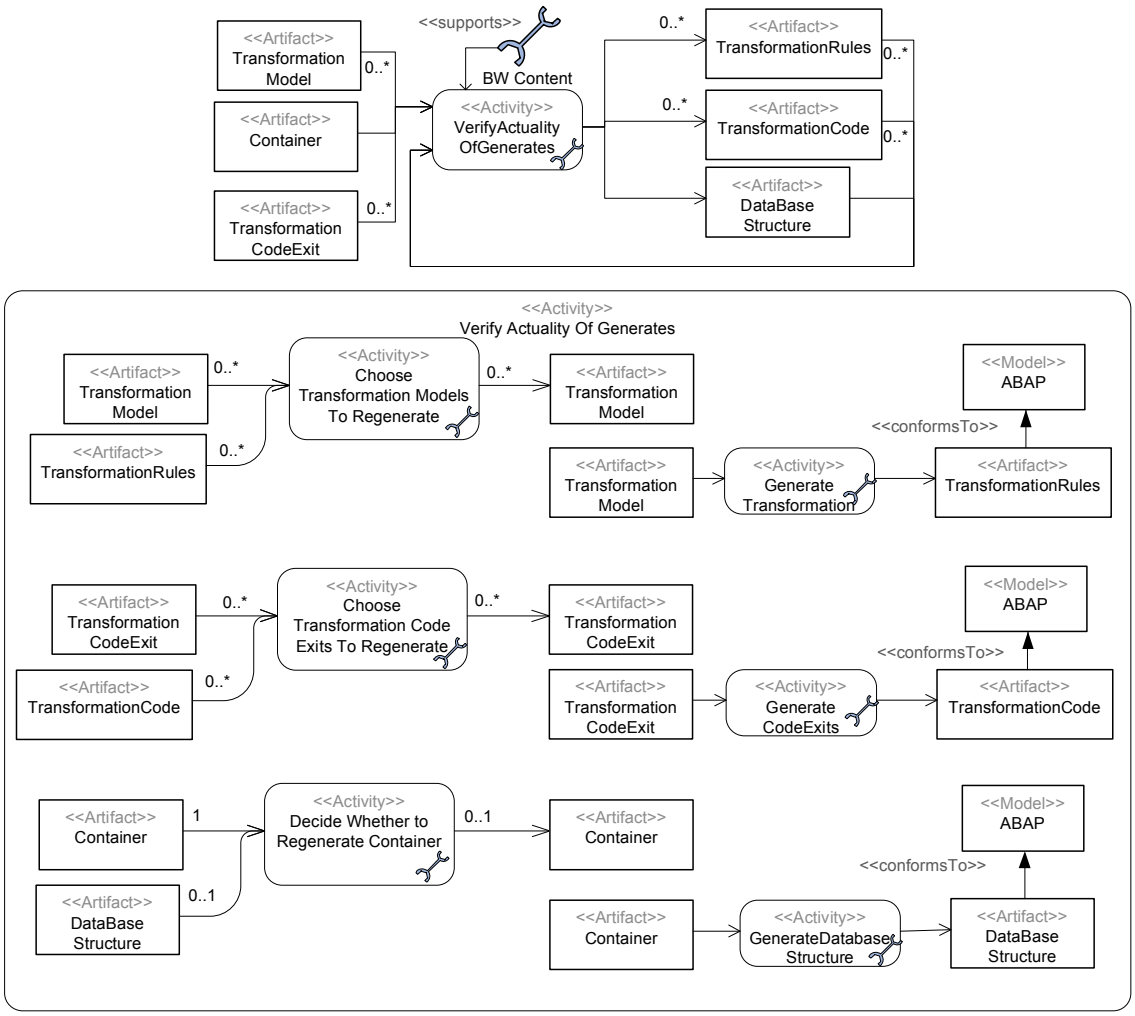

Figure 62: BW: Activity for execution preparation from MDE setting for centralization of reporting using BW

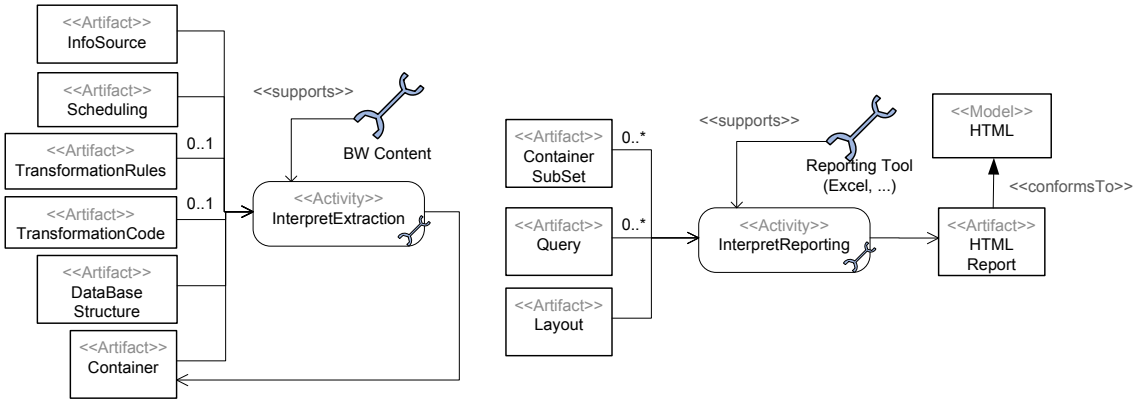

Figure 63: BW: Activities for execution of reporting from MDE setting for centralization of reporting using BW

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
57	978-3-86956-191-2	**Industrial Case Study on the Integration of SysML and AUTOSAR with Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Stefan Neumann, Sebastian Wätzoldt
56	978-3-86956-171-4	**Quantitative Modeling and Analysis of Service-Oriented Real-Time Systems using Interval Probabilistic Timed Automata**	Christian Krause, Holger Giese
55	978-3-86956-169-1	**Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium**	Peter Tröger, Andreas Polze (Eds.)
54	978-3-86956-158-5	**An Abstraction for Version Control Systems**	Matthias Kleine, Robert Hirschfeld, Gilad Bracha
53	978-3-86956-160-8	**Web-based Development in the Lively Kernel**	Jens Lincke, Robert Hirschfeld (Eds.)
52	978-3-86956-156-1	**Einführung von IPv6 in Unternehmensnetzen: Ein Leitfaden**	Wilhelm Boeddinghaus, Christoph Meinel, Harald Sack
51	978-3-86956-148-6	**Advancing the Discovery of Unique Column Combinations**	Ziawasch Abedjan, Felix Naumann
50	978-3-86956-144-8	**Data in Business Processes**	Andreas Meyer, Sergey Smirnov, Mathias Weske
49	978-3-86956-143-1	**Adaptive Windows for Duplicate Detection**	Uwe Draisbach, Felix Naumann, Sascha Szott, Oliver Wonneberg
48	978-3-86956-134-9	**CSOM/PL: A Virtual Machine Product Line**	Michael Haupt, Stefan Marr, Robert Hirschfeld
47	978-3-86956-130-1	**State Propagation in Abstracted Business Processes**	Sergey Smirnov, Armin Zamani Farahani, Mathias Weske
46	978-3-86956-129-5	**Proceedings of the 5th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
45	978-3-86956-128-8	**Survey on Healthcare IT systems: Standards, Regulations and Security**	Christian Neuhaus, Andreas Polze, Mohammad M. R. Chowdhuryy
44	978-3-86956-113-4	**Virtualisierung und Cloud Computing: Konzepte, Technologiestudie, Marktübersicht**	Christoph Meinel, Christian Willems, Sebastian Roschke, Maxim Schnjakin
43	978-3-86956-110-3	**SOA-Security 2010 : Symposium für Sicherheit in Service-orientierten Architekturen ; 28. / 29. Oktober 2010 am Hasso-Plattner-Institut**	Christoph Meinel, Ivonne Thomas, Robert Warschofsky et al.
42	978-3-86956-114-1	**Proceedings of the Fall 2010 Future SOC Lab Day**	Hrsg. von Christoph Meinel, Andreas Polze, Alexander Zeier et al.
41	978-3-86956-108-0	**The effect of tangible media on individuals in business process modeling: A controlled experiment**	Alexander Lübbe

ISBN 978-3-86956-192-9
ISSN 1613-5652