D I S S E R T A T I O N

Supporting Model-Based Reflection, Monitoring, and Evolution in Service-Oriented Architectures through Model-Aware Systems

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.-Prof. Dr. Schahram DUSTDAR
E 184
Institut für Informationssysteme

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Dipl.-Ing. Ta'id HOLMES, DEA
9625914
Argentinierstraße 8/184-1, A-1040 Wien

Diese Dissertation haben begutachtet:

....

Univ.-Prof. Dr. Uwe ZDUN

....

Univ.-Prof. Dr. Schahram DUSTDAR

Wien, am 16. Dezember 2010

....

Dipl.-Ing. Ta'id HOLMES, DEA

Kurzfassung

Die Komplexität servicebasierter Internetsysteme steigt zunehmend, da sie verschiedene Technologien verwenden und konsolidieren, für neue und aufstrebende Technologien adaptiert werden und auferlegten Anforderungen genügen müssen. Viele dieser Systeme sind in Form von präzise spezifizierten Modellen beschrieben; zum Beispiel im Kontext der modellgetriebenen Entwicklung. Während in der modellgetriebenen Entwicklung Modelle hauptsächlich zur Designzeit zum Einsatz kommen, sind wir der Ansicht, dass servicebasierte Systeme besser analysiert und überwacht werden können, wenn Modellinformationen zur Laufzeit zugänglich sind. Zurzeit können modellgetriebene Systeme jedoch nicht auf Modelle zugreifen; z.B. auf Anforderungsmodelle oder aber auf Systemmodelle, von denen sie generiert wurden. Gründe dafür sind fehlende Rückverfolgbarkeitsinformationen nach einem Generationsschritt und Modelländerungen, die eine Koevolution von abhängigen Artefakten und Systemen erfordern. Die Konsolidierung der Modellnutzung und des -managements von Modellen zur Entwicklungs- und Laufzeit wird im Rahmen dieser Dissertation erarbeitet. Hierfür schlagen wir vor, Modelle und deren Elemente eindeutig identifizierbar und in einer verteilten Umgebung abrufbar zu machen und präsentieren modellbewusste Systeme, die die Fähigkeit haben, über Modelle zur Laufzeit zu reflektieren. Für das Monitoring von Anforderungen modellgetriebener Systeme benutzen wir Modellierungstechniken, um Systemanforderungen zu spezifizieren. Wir verbinden diese auf konzeptioneller Ebene mit Modellen, von denen Systeme generiert werden und wenden einen ereignisbasierten Ansatz für modellbewusstes Monitoring, das System- und Anforderungsmodelle in Beziehung bringt, an. In einer industriellen Fallstudie demonstrieren wir, wie das Monitoring von Anforderungen im Compliance Bereich von unserem Ansatz für die Überprüfung von Verletzungen profitieren kann und zeigen, wie modellbewusstes Monitoring die Ursachenanalyse solcher Verletzungen erleichtern kann. Um eine Modellevolution zu unterstützen, studieren wir die Navigationskompatibilität von Modelländerungen, präsentieren einen Algorithmus zur Bestimmung solcher Kompatibilität und stellen eine in Kombination mit Universal Eindeutigen Identifizierern geeignete transparente Modellversionierungstechnik vor, sodass modellbewusste Systeme sich zu jeder Zeit auf ein Modell in einer spezifischen Version beziehen können.

Abstract

Today, service-based Internet systems have become increasingly complex as they comprise and consolidate various technologies, are adapted for new and emerging technologies, and need to comply with imposing requirements. Many of these systems are described in terms of precisely specified models, e.g., in the context of model-driven development (MDD). While models in MDD are primarily used at design time, we argue that by making the information of these models accessible at runtime, we can provide better means for analyzing and monitoring the service-based systems. Currently, model-driven systems are either not aware of their requirements or of the models from which they have been generated. Reasons for this are missing traceability information after a generation step and changes of models that demand the co-evolution of depending artifacts and systems. This, in turn, is impractical for runtime systems that need to operate continuously. This thesis aims at adopting models at runtime, i.e., consolidating design- and runtime use and management of models. For this we propose to make models and model elements uniquely identifiable and retrievable within a distributed environment and present model-aware systems that are capable of reflecting on models at runtime. For the requirements monitoring of model-driven systems we further utilize modeling techniques for specifying system requirements, link these with the models from which systems are generated on a conceptual, modeling level, and employ an event-based approach for model-aware monitoring that relates system and requirement models. In an industrial case study we demonstrate how compliance monitoring can benefit from model-aware monitoring for checking violations at runtime, and show how our approach can ease the root cause analysis of such violations. For supporting model evolution scenarios we conduct a study of the navigation compatibility of model changes, present an algorithm for determining such compatibility, and present a transparent model versioning technique that is suitable in combination with Universally Unique Identifiers, so that a model-aware service can always relate to a model in a particular version.

vouée à

白云

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

千里之行始于足下。

— 老子 [1]

Contents

[1] *A journey of a thousand miles begins with a single step.* — LAOZI

Modern software systems have become increasingly complex. Some reasons for this trend are that software systems often unify different technologies and that their services are exposed to heterogeneous environments. Model-driven engineering (MDE) [23] helps to master the complexity of modern software systems at design time. It utilizes models as central artifacts of the engineering process. In the generation step of the MDE process, models are eventually transformed to source code. This code will be packaged and deployed, but this is where MDE usually stops. This thesis focuses on adopting models at runtime.

This chapter introduces and gives an overview of the context of this thesis. It will point out current research problems and, out of these, extract a set o focal points for this thesis.

1.1 Using Models as Abstractions of Reality

Models are generally used as an abstraction of reality. Stachowiak [152] designates a model to be (1) a *copy* (or representation) and (2) a *shortening* of an original. Also a model has a (3) *pragmatic* function (cf. [11]). Models are commonly used in scientific disciplines (cf. [66]) such as mathematics and the natural and social sciences (cf. [11]). The atom model (cf. [143]) and the molecular model (cf. [14]) in physics and chemistry are examples of *physical* models. Other types of models are *statistical* models, *economic* models (cf. [71]), and *conceptual* models [2].

Models that abstract from an original or from reality can be studied, interpreted, and represented more easily. As they come with a pragmatic function, they are particularly suited for certain purposes (e.g., for representing a system).

1.2 Models in Software Engineering

In many scientific disciplines, particularly natural sciences, models are daily companions, but in the emerging and vibrating field of software engineering (SE) they where gradually developed: In 1976 Chen [37] proposed the entity-relationship model (ERM) for data together with entity-relationship diagrams (ERDs), suitable for database design. His work builds and relates to other models, namely the *network* model (cf. [13]), the *relational* model (cf. [43]), and the *entity set* model (cf. [146, 147, 148]). The object-modeling technique (OMT) (cf. [142]) defined *object* models, *dynamic* models, and *functional* models. It was a predecessor of the Unified Modeling Language [134] (UML), a general purpose modeling language, that was proposed to the Open

[2]Other terms and types of models such as described by Frigg and Hartmann [66] are *scale* models, *idealized* models, *analogical* models, and *phenomenological* models.

Management Group (OMG) in 1997 [3]. It is used in UML diagrams that represent *structural* [4] and *behavioral* [5] views of a system.

View models allow for separation of concerns (SoC) [97], which is one of the successful approaches to manage complexity [70]: the concerns are separated into views that are suitable for capturing and representing the respective concern. The idea of view models has already been applied to software architectures by Kruchten [107] in 1995. His "4+1" view model consists of a *logical* view, a *process* view, a *development* view, a *physical* view, and a *scenarios* view that help different stakeholders of a system "to find out what they want to know about the software architecture" [107, Conclusion].

In order to define the UML the OMG proposed the Meta-Object Facility [131] (MOF) in 2002, a closed meta-modeling architecture for defining meta-models. In 2003 the OMG released a guide for model-driven architecture [130] (MDA) that presents the novel model-driven approach to software design and introduces platform-independent models (PIMs) and platform-specific models (PSMs) from which systems are generated [6]. The idea to not only design a system in terms of models but also generate its software from such a specification is interesting. Finally, models in SE had become prominent: the paradigm of MDE was born.

Today SE can claim to have a couple of generally accepted and internationally standardized models. But with the model-driven approach something special happened as well: Models have been assigned an exceptional function: to actually construct the *objects* they *define*. Thus, in contrast to other models, models in MDE can be regarded as outstanding in the sense that they do not *reflect* but actually *constitute* originals.

1.3 Fostering the Use of Models

Nowadays, there is a trend toward the use of precisely specified models, for example, for model-driven development (MDD) [176]. Some reasons for using MDD include: Instances of the models can be validated for specified properties; Models can be defined and refined at different abstraction levels; This makes the models suitable to be used by various stakeholders, e.g., domain experts who use high-level, graphical models and technical experts who work with more low-level, textual models [125, 129]; Technical expertise can be captured in transformations,

[3]Its version 1.4.2 [94] became an ISO standard in 2005. Also version 2.1.2 [96] is in the standardization process. The most recent version from 2010 is 2.3.

[4]i.e., class, component, composite structure, deployment, object, package, or profile diagrams

[5]i.e., activity, state machine, use case, or interaction (i.e., communication, overview, sequence, or timing) diagrams

[6]A PIM is a model of a software system that does not depend on the specific technologies or platforms used to implement it while a PSM links to particular technologies or platforms.

e.g., when PIMs are transformed to PSMs; This enhances portability and simplifies adaptations; Recurring code can be generated, easing the maintenance of a model-driven system.

Because of these advantages, we can express a motivation for fostering the use of models. We argue that while models can be used for describing and designing the system and its domain, models can also be used at runtime. This would be worthwhile because models have precise semantics and can be used for communication between the different stakeholders. Also we argue that in a couple of scenarios such as service monitoring or adaptation (cf. [38]) it would be desirable to have access to models (e.g., models from which the system has been generated). For example, this can help to display or analyze the models that cause exceptional situations in the system, or to react to such situations using a manual or automated adaptation of the system. In this way, *stakeholders* can relate to the concepts of a model more easily. In addition, *services* of the system may profit from introspecting models at runtime. We will demonstrate our idea of expanding the usage of models from design time to runtime in a motivating scenario in Section 1.5.

1.4 Challenges

In the broader view of model-driven systems in general, common problems in MDD systems are collaboration, traceability, and model evolution. In this section, we want to motivate and explain briefly how our approach helps to address these issues. We describe these common problems also to set the scope for this thesis and delineate which parts of the general problems are addressed by our approach and which are not.

1.4.1 Support for Collaboration in Model-Driven Systems

Most current tool support for MDD only focuses on the design time (cf. [175]) and comes with limited collaboration features, if any. Model-aware services that we will introduce in this thesis, however, rather assume a distributed environment, maybe even distributed development. In order to facilitate various services of a distributed environment, to concurrently work with MDD projects and artifacts we need to support the management of projects and artifacts, with (1) versioning capabilities while capturing and keeping track of model relationships; (2) services for the information retrieval (IR) and the management of these; For (3) facilitating collaboration scenarios, we also need to (4) deal with concurrency, e.g., provide locking mechanisms, raise the awareness for the work of others; offer compare and merge possibilities (cf. [9]) as well as support for resolving conflicts (cf. [31]). To support various stakeholders appropriate model-representations are required in form of domain-specific languages (DSLs). For security, role-based access controls (RBACs) can be applied to MDE artifacts.

This thesis covers the first two of these issues, versioning capabilities (cf. Section 3.4) and services for IR and management (cf. Section 3.5). These features are also needed for the monitoring, auditing, reporting, and business intelligence purposes (cf. Chapter 6) addressed by the model-aware services approach (cf. Chapter 4) proposed in this thesis. For example, a monitoring component requires a service for retrieving model information and it requires the models in the version of the model instance that it monitors. Please note that the collaboration features could also be used for other scenarios, such as supporting the MDD process for distributed development. While we do not neglect the importance of collaboration in the context of MDD, this thesis has a particular focus that is positioned at the runtime of MDE – not at the design time. Having said that, this work is not limited to the runtime but rather aims at bridging the gap between design time and runtime in terms of model use and management.

1.4.2 Traceability in Model-Driven Systems

A particular problem of model-driven systems is traceability – asking the question: How do models and model elements of different abstraction levels and/or code correspond to each other? In particular, the traceability information for models that are transformed into other models or code can get lost in model-driven approaches. On the one hand, a transformation rule describes how source models are mapped to target models; on the other hand a traceability link to a target model would allow for identifying the source models. Traceability is essential for meaningful feedback from the runtime to stakeholders and for identifying and understanding the root cause, e.g., in case of a failure or exception. Obviously, if we are able to trace the source model from which a target (model or code) has been created (via generation or transformation), it is possible to then use the information in the source model to analyze or debug the target.

Aizenbud et al. [5] discuss model traceability in the context of MDD. Among their suggestions are to establish a standard traceability meta-model and to uniquely identify artifacts across space and time which is typically not supported by the current toolchain [5]. The latter is within the scope of our work as such unique identification is required for the runtime use of models, in particular when these are subject to change. For the traceability of model-driven services (see Section 4.1) we propose to (1) embed traceability information automatically and (2) expose this information in a service-oriented fashion. A *service* is in the first place a distributed object that is accessible via the network and has certain characteristics: The service offers a public interface and is both platform- and protocol-independent. It is self-contained in the sense that it can interdependently be used, and no implementation details need to be known for using the service. Service-oriented architecture (SOA) is the main architectural style for service-oriented computing (SOC). In Listing 4.1 we further propose a traceability matrix that is particularly suitable for the annotation of process-driven SOAs (see also Section 4.2). In a *process-driven SOA*, a process engine is used to orchestrate the services in order to implement business processes [80].

1.4.3 Model Evolution

The broad adoption of model-driven approaches, e.g., for the development of software systems, resulted in an increasing importance of models as central artifacts. In the context of MDD software systems are generated from models. Because such systems need to be maintained and evolve in order to address new system requirements, also the respective models need to evolve likewise.

Sometimes, for example in case of maintenance, changes are minor. During an evolutionary step, however, changes are major. But a new version of a model may still have to be compatible with the original version. A common problem when maintaining and evolving software systems is the compatibility of the changed system and its original: is the new version compatible with the old version? Particularly we would like to know if two models are navigation compatible. That is, we would like to compare two models in a sense, so that we can deduce if they are compatible in regard to the original navigation. In the context of MDD where software systems are generated from models which in turn conform to meta-models, this question is especially interesting. If we can facilitate such a check at the modeling level (e.g., for changes that occur to meta-models) this will also be a step towards ensuring the compatibility of software systems (cf. [62]). We will address this particular problem in the area of model evolution in Chapter 2.

1.5 Motivating Scenario

Often it is hard to manage, analyze, and monitor complex service-based systems at runtime, because (1) the complexity of service-bases systems needs to be mastered and (2) requirements that are imposed on such systems have to be monitored. For various reasons, many systems (cf. [123, 186]) and requirements (cf. [150, 41]) today are modeled with precisely specified and detailed models. One of these reasons is the increasing use of MDD that helps to master the complexity of systems during development. Management, analysis, and monitoring results could be improved by making the information in the precisely specified models accessible at runtime.

Let us thus give a motivating scenario that exemplifies a general setup and will act as the starting point for further discussion. Thus, we depict the context and introduce the prerequisites of our work. In this scenario, MDD is applied in order to develop services of a SOA. Therefore, these services relate to models. At runtime a monitoring service (or monitor for short) first receives notifications from the model-driven services of the SOA on invocation and termination. This is realized via eventing. Hence, the services emit events and the monitor processes these (cf. [48] for service level agreement (SLA) monitoring). Note that in our scenario the events *relate to models*. That is, if a service is invoked, it emits an event with an identifier of the model from which it has been generated. Also please note that this is a simple, *generic* setup that can be extended if necessary. That is, if the monitor needs to receive more events than just invocation and termination, or if it needs to consider additional information, this would be provided by the model-driven services, e.g., by generating and embedding the required eventing into the services.

In a next step, the monitor can use the information from the models to which the events relate. For example, these can be the models from which the services have been generated. In a more elaborate setup, such events can also relate to other models as well. Next, for facilitating the governance, management, and adaptation of the SOA in such a scenario the monitor needs to consider the information from the models. Typically, this is done by reflecting on the value or state of model elements to which certain semantics have been attributed. For example, the monitor can compare values against thresholds. Ideally, the monitor will be able to retrieve relevant models dynamically at runtime. With the results from inspecting the various models, all the necessary information is available to the monitor for reporting, realizing governance tasks, or initiating an adaptation.

A monitor itself relies on configurations (e.g., thresholds). We argue that configurations are captured ideally in the form of models, too (cf. [38] for goal models). This allows for the generic use of models at runtime. Besides retrieving such configurations, a monitor for adaptation typically also operates on models that are specific to the reporting, governance, or adaptation. Such models can contain data from the monitoring such as aggregated statistics. Thus, the monitor not only retrieves models at runtime but also operates on models.

There are a couple of obstacles prior to the adoption of models in SOAs. First, in a distributed and heterogeneous environment the identification and retrieval of models from services is challenging. This is because of the need of unique identifiers across the SOA and the lack of a common, unified, and service-based access to models. Second, if a meta-model is modified on which a given monitor depends the monitoring service also has to be changed; i.e., it usually has to be manually adapted to work with the new version. This implies a manual maintenance of the source code, recompilation, testing, and redeployment of the monitoring service. Such evolution steps thus impose manual effort and this in turn impedes the use of models at runtime. Because this involves manual effort, adopting models without automation support becomes not only expensive but also impractical. Beyond that, once a service model is used by autonomous services in a SOA, the service management must consider model evolution. That is, while new versions of service models exist, old versions need to be supported and the various services in a SOA need to relate and work with specific, coexisting versions.

1.6 Research Questions

In this section we summarize this thesis' main research questions:

Artifacts, such as meta-models in model-driven engineering, are subject to change. Evolution of meta-models may require co-evolution of artifacts and systems if navigability is affected: How can navigation incompatibilities be detected? Systems may dynamically, i.e., at runtime, reflect on models. If a change is introduced to a meta-model, to which models conform, this may break systems. For this reason we would like to identify navigation compatible model changes that – however limiting such changes may be – allow to change continuously running model-driven systems. By answering this question we introduce the novel concepts of navigation compatible models and navigation compatible model changes.

During execution, systems could benefit from reflecting on models. For adopting models at runtime: How can model-driven engineering projects and artifacts – particularly models and model elements – that are subject to change, be retrieved, searched for, and managed both at design time and runtime in a distributed setting with services and developers involved? Model-aware entities (i.e., system or human users) consume, produce, or reflect on MDE artifacts such as models. This may take place during design time but can also happen at runtime. We have developed a conceptual model (for the management) of MDE projects that is the base constructive model for the Model-Aware Service Environment [82, 87, 86] (MORSE) repository, which enables model-aware entities to concurrently work with MDE artifacts at design time as well as at runtime in a distributed, service-based environment.

Monitoring could be enhanced if conceptual models such as requirement and system models would be considered and related at runtime: How to facilitate root cause analysis of runtime violations at the abstraction level of models? For this we relate runtime systems with (1) models from which they have been generated and (2) their imposed requirements that may be subject to change. With our preparatory work, i.e., model-aware system, we are able to support the monitoring at the abstraction level of models that we call model-aware monitoring. We will demonstrate the root cause analysis of runtime violations of process-driven service-oriented systems in an industrial case study.

1.7 Scientific Contributions

By addressing the research questions previously mentioned, this thesis achieves the following scientific contributions:

- Given the continuously growing adoption of MDE practices and the rising complexity of service-based systems, we show the usefulness of shifting the focus of model management and use from design time to runtime.
- By applying a MDE approach while adopting models at runtime, this thesis demonstrates a *novel*, direct linkage and correlation of system & requirement models.
- In particular, by coupling model-driven systems with event-driven architectures (EDAs) at runtime, we enable dynamic, model-aware monitoring. That is, the monitoring can take place at defined levels of abstraction as reflected by the models.
- For this we present *novel* model-aware services and processes that contain, emit, or expose model traceability information that can be used for model look-up and reflection during runtime.
- We define the *novel* notion of a navigation compatible model change and present an algorithm for determining navigation compatibility that can be applied to facilitate continuous evolution of models during MDE.
- Hiding the complexity of implicit versioning of models and model elements from users while respecting the Universally Unique Identifier [113, 98] (UUID) principle, we present a *novel* transparent UUID-based model versioning technique.
- In a distributed, service-oriented environment we make MDE artifacts, such as models and model elements, uniquely identifiable and automate the generation of retrieval and management services.
- By applying a service-oriented and UUID-based approach we unify the use and management of MDE projects and artifacts for design time and runtime clients.

1.8 Structure of Dissertation

This thesis is structured as follows:

Chapter 2 Models are subject to change; thus, in an evolving environment and for the management of model-driven systems we need to study and understand the implications of such changes. In this chapter, we define and focus on one type of change: navigation compatible model changes.

Chapter 3 For adopting models at runtime we will present our novel MORSE approach. We will propose a model-driven approach for the generation of service-oriented model repositories that bridge the gap between design time and runtime use and management of models. While being suitable for design time tools, MORSE primarily targets runtime systems, makes artifacts universally and uniquely identifiable and applies a transparent UUID-based model versioning.

Chapter 4 With the established MORSE foundation for adopting models at runtime we will then propose model-aware services and demonstrate how business processes, i.e., process-driven SOAs, can be integrated with such model-aware services.

Chapter 5 Dynamic reflection on models is interesting in a monitoring and adaptation context, e.g., the monitoring of system requirements. However, before we can focus on such monitoring we first need to realize a connection between systems and system requirements on a conceptual modeling level. Thus, in this chapter we will demonstrate how to model and link these in the context of business compliance through the power of MDE.

Chapter 6 In this chapter we will concentrate on model-aware monitoring, a consecutive contribution that builds on MORSE, model-aware services, and the view-based modeling of systems and requirements. With model-aware monitoring the monitoring of runtime systems can take place at the model level. This chapter focuses on the runtime aspects of process-driven service-oriented systems and closes a feedback loop from the MDE runtime to the design time. In a case study we demonstrate this for the compliance monitoring of business processes.

Chapter 7 Finally, we summarize our contributions and conclude with a prospect of further work.

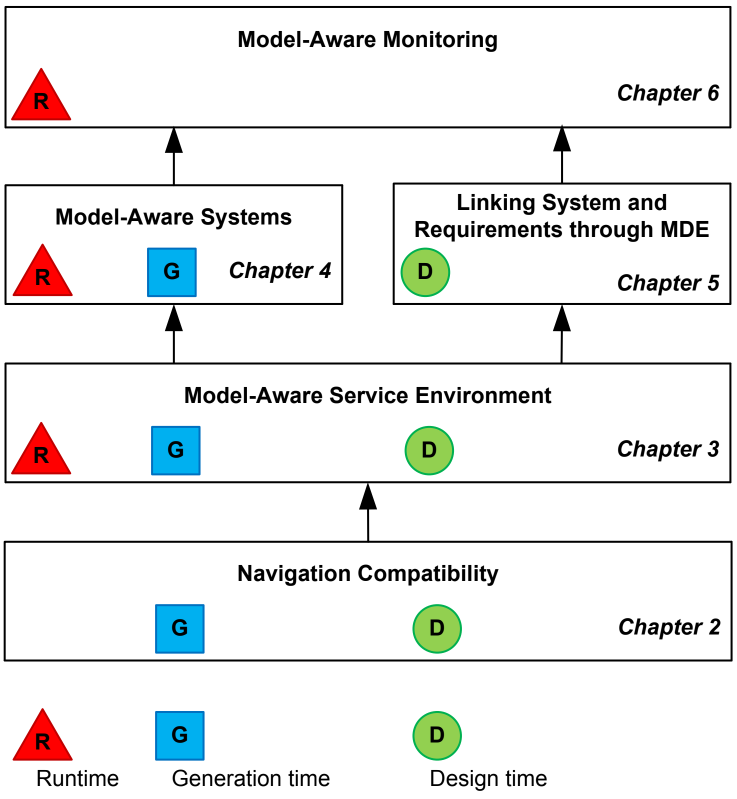


Figure 1.1: Dependency Graph of the Dissertation's Chapters

The dependency diagram in Figure 1.1 gives an overview of dependencies of the the dissertation's chapters. Tags indicate if the chapter focuses on the design time, deployment time or runtime. From the diagram we can see that, e.g., Chapter 3 – that covers design time, deployment time, and runtime – constitutes the foundation for the chapters on model-aware services – that has a focus on the deployment time and runtime – and Chapter 5 that covers design time aspects.

CHAPTER 2

Navigation Compatibility of Models

Ἐν πᾶσι γὰρ τοῖς φυσικοῖς ἔνεστί τι θαυμαστόν.
— Ἀριστοτέλης [1]

Contents

[1] *In all things of nature there is something of the marvelous.* — ARISTOTLE

Changes to models, for instance in the context of model-driven development (MDD), can result in incompatibilities. For example, a model-to-code transformation template may not be able to process a new version of a changed meta-model. Ideally, a developer who operates on a model would receive some early feedback from a system in case of incompatibilities prior to a code generation step. In this chapter we focus on the navigation compatibility of models and discuss different modifications that can occur to models and their impact in the context of MDD. Our approach, inspired by the idea of transitive reasoning, considers roles and multiplicities and distinguishes between categories of navigation compatible relationships. In this chapter we present an algorithm that determines the navigation compatibility of two models. Using an original and modified version of a model this algorithm can be applied for checking the navigation compatibility of a model change. It provides detailed feedback that can help developers to identify and adjust navigation incompatibilities. Prior to the algorithm we present a detailed study that builds on previous work from transitive reasoning and contributes a set of novel conclusions for not yet covered aspects such as navigability, roles, and multiplicities. Finally, we evaluate the run-time performance and provide results from analyzing and applying our findings to model-driven projects.

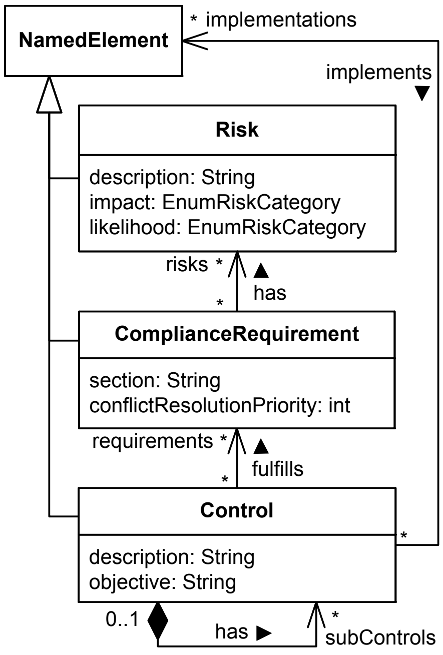
This chapter is structured as follows: In the next section we introduce the context and focus of our work with a motivating example. For outlining the problem of navigation compatibility, the succeeding section illustrates some model changes. Next, Section 2.3 introduces the idea of transitive reasoning for reducing a changed model to its original and discusses the approach in the context of MDD. We then present our algorithm for checking the navigation compatibility in Section 2.4. Section 2.5 discusses results from analyzing a MDD project, presents the runtime complexity of the algorithm as well as performance results from industrial and open-source meta-models, and closes with a discussion on lessons learned and experiences in an industrial setting. Finally, in Section 2.6 we compare to related work and we conclude in Section 2.7.

2.1 Motivating Example

In this section we illustrate the problem of navigation compatibility of model changes with a motivating example. While our work can be applied on any models and model instances of the same abstraction, the focus of this chapter lies on comparing meta-models in the $M2$ level (cf. Open Management Group (OMG) definition [130]). Note that the term *model* is generally applicable for any model level. For simplicity we continue to use the generic term *model* and mean the meta-model $M2$ level unless otherwise mentioned.

We focus on such models because in a model-driven context they are used in model transformations. That is, instances of such models are transformed to code in case of a model-to-code transformation. Using model-to-model transformations they can also be transformed to instances that conform to other models. For the development of a model transformation not the instances but only the models (i.e., meta-models) are required. For example expressions such as Object Constraint Language [133] (OCL) expressions that are used in transformation languages operate on the model. *Navigating expressions* are a subset of such expressions that navigate through relations.

Figure 2.1a displays an excerpt of a model from the domain of business compliance that is applied by a European mobile virtual network operator (MVNO) (see also Section 2.5.5). The company uses this model with a model-to-code template displayed in Figure 2.1b for generating documentation for a web page. In the model concepts such as `Control`, `Compliance-Requirement`, and `Risk` are defined that are used by the template. The latter contains an navigating expression `c.requirements.risks.contains(r)` that checks for a control `c` if it is associated with a specific risk `r` through a requirement. That is, the expression navigates from the control *via* requirements to risks.



(a) Excerpt of a Compliance Model

```
<h2>Risk-Control Correlation Matrix</h2>
<table>
  <tr>
    <th>Risks/Controls</th>
«FOREACH cv.control AS c»
    <th>
      <a href="«cv.processName+
        "_C_"+c.uuid+".html"»">«c.name»</a>
    </th>
«ENDFOREACH»
  </tr>
«FOREACH cv.risk AS r»
  <tr>
    <th>
      <a href="«cv.processName+
        "_R_"+r.uuid+".html"»">«r.name»</a>
    </th>
  «FOREACH cv.control AS c»
    <td>
    «IF (c.requirements.risks.contains(r))»
      X
    «ENDIF»
    </td>
  «ENDFOREACH»
  </tr>
«ENDFOREACH»
</table>
```

(b) Model-to-Code Transformation Template

Figure 2.1: Meta-Model and Model Transformation Examples

In case a model is changed usually also transformations need to be aligned. For example, if the risks would be directly associated with controls the navigating expression would have to be simplified to `c.risks.contains(r)`. In this chapter we focus on changes that affect the *navigation compatibility* of model changes (see also Definition 2 in the subsequent section). In our example a navigation compatible change would not require an alignment in *navigating expressions*, e.g., in a model transformation, whereas a navigation incompatible change would. In the latter case it would be desirable to obtain detailed information on the incompatibilities so that an alignment can be better supported.

2.2 Navigation Compatibility

In this section we first define the notion of a navigation compatible model with Definition 2 and illustrate the problem of navigation compatibility with some navigation compatible model changes [2].

Definition 1 *Let f be a bijective mapping function $M \rightarrow M'$ between model elements $m \in M$ and $m' \in M'$ that have the same model element type.*

The mapping function from Definition 1 that we use in our definition for navigation compatibility either maps identical or renamed model elements and thus supports the renaming of model elements.

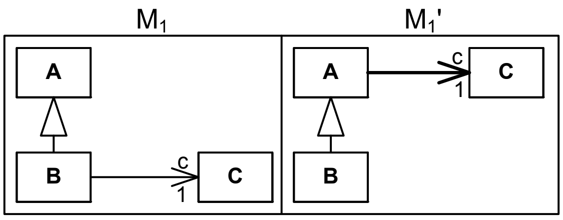
Definition 2 *A model M' is navigation compatible to a model M if $\forall$ concrete classes $c \in M$ there $\exists$ a class $f(c) \in M'$ and if $\forall$ references $r \in c$ referencing a class $d \in M$ there $\exists$ a reference $f(r) \in f(c)$ that references a class or subclass of $f(d) \in M'$.*

For further introducing the problem of navigation compatibility let us start with a few examples: Figure 2.2 shows two versions of models with the original versions on the left and the modified versions on the right.

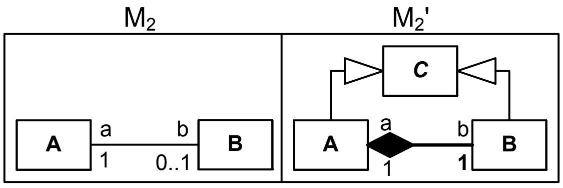
In our first example (Figure 2.2a) a reference to `C` was moved from `B` to the superclass `A`. After the change `B` can still navigate to `C` via the inherited relationship using role `c`. Note that the relationship in M_1 is not navigable from `C` to `B`. If it would be, such a modification would result in a navigation incompatibility, i.e., while a new navigation from `C` to `A` would be introduced, the navigation from `C` to `B` would be missing in M_1'.

Another navigation compatible model change is illustrated in Figure 2.2b. First, the multiplicity of role `b` was modified. Second, the type of the relationship was changed from association

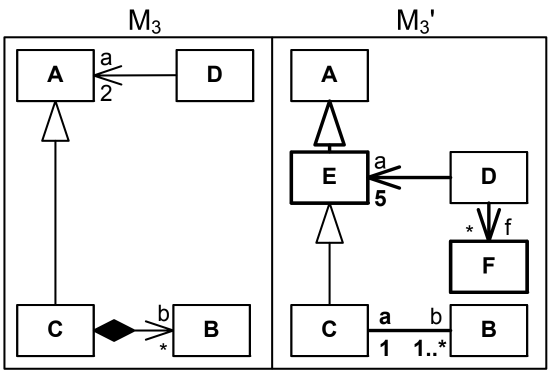
[2]Generally, we adopt terminology from the Meta-Object Facility [131] (MOF) [131] and Unified Modeling Language [134] (UML) [134]. With *concrete* classes we mean classes that are not abstract. We further use the term *reference* as an element of a class for (not only) uni-directional but also bi-directional relationships: in the latter case both classes of the relationship ends contain a reference to the other class.



(a) Relationship moved to Superclass



(b) Change of Relationship Type and Multiplicity and Introduction of an Abstract Superclass



(c) Extended the Object Hierarchy and Applied Various Changes

Figure 2.2: Examples of Navigation Compatible Model Changes

to composition. Third, a common, abstract superclass `C` was introduced. Still, the navigation from `A` to `B` and vice versa is possible as in the original version.

Finally, Figure 2.2c depicts an example of a more complex, navigation compatible, model change that contains various modifications to the model. Note that some of these modifications resemble the presented ones from Figure 2.2a and 2.2b. For example, similar to M_1 the relationship from `D` to `A` in M_3 was moved to a different class in the inheritance hierarchy. As in M_2, the type of the relationship between `C` and `B` was changed . Finally, new classes such as `F` and additional relationships such as from `D` to `F` may be added to the model without affecting the original navigability.

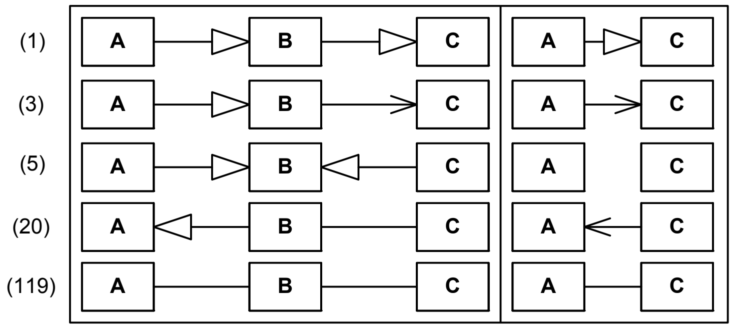


Figure 2.3: Some of the 121 Reduction Rules from Egyed [58]

2.3 Transitive Abstraction Rules

In the previous section we have given examples of navigation compatible modifications and with Figure 2.2c we have grasped the idea that a navigation compatible model change can consist of various simple navigation compatible modifications. If we find a model change to consist of multiple navigation compatible modifications we can deduce from that that the overall model change is navigation compatible. Pursuing this idea, this lead us to the following question: Is it possible to reduce a changed model to its original using transitive abstraction? This problem is covered by the literature, e.g., Egyed [58, 59] presents 121 transitive abstraction rules for transforming “low-level” models to more abstract, “higher-level” models as depicted in Figure 2.3. Thus, transitive abstraction reduces the complexity of a model by removing model elements according to transitive reduction rules.

If we look at our examples, we find that we can apply Reduction Rule 3 on M_1' in order to obtain the original relationship between `B` and `C` of M_1. Similarly, M_2' can be reduced by applying Rule 5. Reduction Rule 20 can be applied on M_3' before Rule 1 simplifies the transitive inheritance relationship between `A`, `E`, and `C` into the relationship between `A` and `C` as found in M_3.

While this approach is promising for our purposes of determining the navigation compatibility of a model change, the presented work does not examine navigation compatibility in transitive abstraction. Therefore it is not applicable for MDD. Particularly, *navigability*, *roles* and *multiplicities* are not (sufficiently) covered by the work. As a consequence, some rules are not applicable within the context of MDD. Having said that, many of the rules can be grouped into categories for the purpose of examining the navigation compatibility. In the following we will refer to conclusions of our study that we list in Section 2.3.5.

2.3.1 Navigability

Within the context of MDD we cannot simply apply the rules from [58]: E.g., Rule 119 simplifies the relationship of class `A` to class `C` via class `B`: An indirect relationship becomes a direct relationship between the two classes. However, navigability cannot be simplified in MDD as such: e.g., in a model-to-code template `C` cannot be accessed from `A` directly but needs to be navigated via `B` (see also our motivating example from Section 2.1). Such a navigation incompatible change in the model would necessitate a change in the template. As a consequence, associations must not be reduced via transitive rules for navigation compatibility (Conclusion 2). Besides indirect relationships we also need to look at the navigability of a single relationship. A relationship can change from a relationship with a single navigable end to a bidirectional relationship with two navigable ends as represented by the model change Δ_A in Figure 2.4 (Conclusion 3). This modification is also found in the model change in Figure 2.2c between the classes `B` and `C`: a new role `a` is introduced with a multiplicity of `1`. Note that we will discuss roles and multiplicities below.

A relationship end can be explicitly marked as unnavigable. This semantic may – in case the relationship is unidirectional – be added or removed (Conclusion 4) (see also Δ_B and Δ_C in Figure 2.4). If both ends are explicitly marked as unnavigable and we would like to remove the semantic on one end this end also has to become navigable (Conclusion 5) (see also Δ_D in Figure 2.4). By applying multiple rules we can draw further conclusions: e.g., that ends that are explicitly specified as unnavigable can become navigable. Last but not least, for navigation compatibility, obviously, a navigable end must stay navigable after a model change (Conclusion 6).

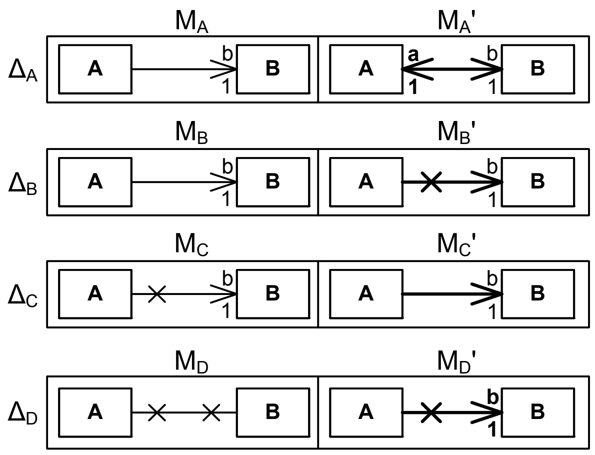


Figure 2.4: Navigation Compatible Changes to the Navigability of Relationship Ends

2.3.2 Roles

Navigation over a reference is realized by a role. The role name is important within the context of MDD, e.g., templates hard code roles in order to access referenced objects that act according to a role. For example, if multiple references from one class `A` to another class `B` exist, the object can distinguish referenced objects via role names. Thus, role names must not change (Conclusion 7). Nonetheless, renames can be supported (see Definition 1 and Section 2.5.4).

2.3.3 Relationships

In some cases the type of the relationship may change without affecting the original navigability of the model. For example in Figure 2.2b and Figure 2.2c we have seen how an association can be changed to a composition and vice versa.

We distinguish two categories of relationships: First, generalization and realization. Second, association, aggregation, and composition. The relationships within these categories are similar in a sense that we may modify the type of the relationship within such a group with the derived models to be navigation compatible (Conclusion 8). Contrary, if the new and old type of a changed relationship are from different categories of relationships the change is navigation incompatible.

2.3.4 Multiplicities

Another aspect that should be considered in the MDD context is the multiplicities of relationship ends. That is, while multiplicities are mentioned in the work from Egyed, the discussion is limited to transitive abstraction. As with the navigability of relationship ends (cf. Section 2.3.1) we also need to examine the modifications on the multiplicities of relationship ends. Multiplicities are relevant in this discussion because, usually, different multiplicities are handled differently. For multiplicities with an upper bound greater than one a transformation rule in a model-driven template typically iterates over the instances (using a `foreach` statement) [3]. For the case of optionality (i.e., multiplicities with a lower bound equal to zero), typically a condition (using an `if` statement) checks for the existence of objects [3]. Table 2.1 specifies how relationships with different multiplicities are usually dealt with in transformation and programming languages. For each possible multiplicity an example is given for a typical statement. In these statements `b` refers to a role with a role name *b*.

Note that our examples from Figure 2.2 also contain some relevant modifications: in Figure 2.2b the multiplicity of role `b` was changed from `0..1` to `1`, Figure 2.2c contains two mod-

[3]see also the model-to-code template in Figure 2.1b

Table 2.1: Processing Relationships with Different Multiplicities

Multiplicity	**Constraints**	**Statements**
1		`self`.b
0..1		if `self`.$b \neq \emptyset$: `self`.b
x	$x > 1$	foreach e in b
0..x	$x > 1$	if `self`.$b \neq \emptyset$: foreach e in b
x..y	$x \geq 0, y > x$	foreach e in b

ifications of the multiplicity of the roles `a` and `b`. Both examples eliminate optionality. This is a navigation compatible modification (Conclusion 9) as the template would not have to be changed: the condition that checks for the existence will successfully evaluate and execute the statement as in the case without optionality.

The various modifications are summarized in Figure 2.5 and the Conclusions 9-13. When enforced they do not involve the necessity to align navigating expressions (see Section 2.1).

An ancillary finding of this examination is the use of iterations in transformation templates: Supposed the transformation language's `foreach` statement could operate on single elements in addition to sets it would be preferable to always use such a statement as in case of changed multiplicities the transformation would not have to be changed. For the optionality and the conditional check for existence, languages could similarly combine a check with the iteration of one or more elements in a general applicable statement. Such a statement, if applied in preference to the other statements, would have the advantage that, in case multiplicities of models change, templates would not have to be modified.

2.3.5 Conclusions

The following list summarizes the conclusions from our examination for navigation compatible changes:

Conclusion 1 *New model elements such as classes and/or additional relationships* MAY *be introduced with a model change without affecting the original navigability.*

Conclusion 2 *Rules that reduce associations* MUST NOT *be applied for transitive navigation compatibility.*

Conclusion 3 *An unidirectional relationship* MAY *be changed to a bidirectional one.*

Conclusion 4 *The semantic of a relationship end to be unnavigable* MAY *be added or removed in case of an unidirectional relationship.*

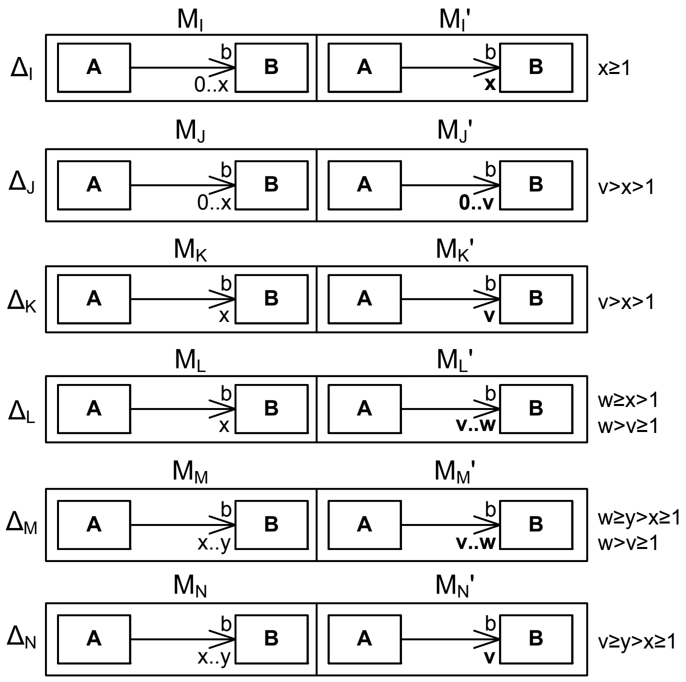


Figure 2.5: Changes to Multiplicities that do not break Navigating Expressions

Conclusion 5 *An end of a relationship with two unnavigable ends* MAY *become navigable.*

Conclusion 6 *A navigable end* MUST NOT *be rendered unnavigable.*

Conclusion 7 *For navigation compatibility role names* MUST NOT *change.* [4]

Conclusion 8 *A relationship of type generalization* MAY *be changed to a realization relationship and vice versa. Also, association, aggregation, and composition relationships are interchangeable.*

Conclusion 9 *The optionality of a multiplicity* MAY *be removed.*

Conclusion 10 *A change in the multiplicity* SHOULD NOT *introduce optionality.*

Conclusion 11 *A multiplicity of one* SHOULD NOT *be changed.*

Conclusion 12 *A multiplicity of* `0..1` SHOULD NOT *be changed to a multiplicity other than* `1`.

Conclusion 13 *The upper bound of a multiplicity* SHOULD NOT *be reduced.*

[4]We can relax this restriction if we consider renames (see Definition 1 and Section 2.5.4).

In summary and according to our definition, a navigation compatible change is a *precondition* for not having to co-evolve navigating expressions. Changes that SHOULD NOT apply *may* imply the need for co-evolving navigating expressions, still according to our definition for navigation compatible models in Section 2.2 such changes are navigation compatible. This particularly concerns changes to multiplicities (see also Section 2.3.4 and Table 2.1) that may require adaptation according to the language in which navigation expressions are formulated. Our algorithm for checking the navigation compatibility that we present in the next section will issue warnings in these cases for supporting an alignment.

2.4 Automated Navigation Compatibility Checks

In this section we present an algorithm [5] that can be applied for determining the navigation compatibility of a model change. The input parameters for Algorithm 1 are two models, e.g., an original version m and the modified version m' of a model change. Besides a logical result that signifies the navigation compatibility of m' against m, the algorithm may return hash tables with `warnings` and `errors` in addition. These hash tables can contain information on a missing class or relationship or warn about changed multiplicities. As a key the class name of m in question or the name of the role (with the class name and a dot as prefix) is used. This information can serve as valuable feedback to a modeling tool and the user for interactive development.

After the existence of classes and references (covering Conclusion 7) have been ensured, Algorithm 2 continues to verify the relationships (called at Line 20 of Algorithm 1). Please note that by considering all generalizations and realizations (cf. Line 3-5 and 7 of Algorithm 2), this part of the overall algorithm covers the *various* transitive reduction rules as relevant for our purpose of navigation compatibility such as Rule 1, 3, and 20 from Figure 2.3. This simplifies the algorithm, shortens the runtime, and thus improves performance. Note that the algorithm allows a model change to introduce new model elements (covering Conclusion 1).

Besides the mentioned relationships, the algorithm makes use of *references* for verifying the navigability of relationships. A reference from a class `A` to `B` represents an unidirectional relationship with navigability from `A` to `B`. The relationship type of a reference can be association, aggregation, or generalization (covering Conclusion 8). A bidirectional relationship between two classes is realized with two opposing references. Thus, the model change Δ_A from Figure 2.4 results in a new reference from `B` to `A` (covered by Conclusion 1). The same applies for

[5]The formalism to describe the algorithm is derived from the Frag language [183, 184]. Particularly, statements that operate on hashtables are expressed as `$hashtable set $key $value` and `$hashtable get $key`. Note that the algorithm makes use of some additional helper functions that are not shown for brevity reasons such as `getSuperclasses($class)`.

Algorithm 1: Navigation Compatibility Check

```
   Input: $m, m' \in M$
   Output: compatible $\in$ {true, false}, warnings, errors
1  begin
2      warnings, errors, className2class' $\longleftarrow \emptyset$;
3      for class' $\in$ getClasses(m') do
4          $className2class' set getName($class') $class';
5      for class $\in$ getClasses(m) do
6          className $\longleftarrow$ getName($class);
7          class' $\longleftarrow$ $className2class' get $className;
8          if $class' $= \emptyset$ then
9              $errors set $className "missing";
10         else
11             role2reference' $\longleftarrow \emptyset$;
12             for reference' $\in$ getReferences($class' $\cup$ getSuperclasses($class')) do
13                 $role2reference' set getName($reference') $reference';
14             for reference $\in$ getReferences($class $\cup$ getSuperclasses($class)) do
15                 role $\longleftarrow$ getName($reference);
16                 reference' $\longleftarrow$ $role2reference' get $role;
17                 if $reference' $= \emptyset$ then
18                     $errors set "$className.$role" "missing";
19                 else
20                     error $\longleftarrow$ Generalization Check ($reference, $reference');
21                     if error $\neq \emptyset$ then
22                         $errors set "$className.$role" $error;
23                     else
24                         for warning $\in$ Multiplicity Check ($reference, $reference') do
25                             $warnings set "$className.$role" $warning;
26     $\longleftarrow$ {compatible $\longleftarrow$ $errors size = 0, warnings, errors};
27 end
```

Δ_D. The other changes from Figure 2.4 (in order to cover the Conclusions 2-6), Δ_B and Δ_C, do not affect the navigability from `B` to `A` and do not have to be identified by the algorithm.

Algorithm 2: Generalization Check

```
Input: reference, reference′
Output: error
begin
    rClass′ ⟵ getClass($reference′);
    allRClassifierNames′ ⟵ ∅;
    for rClassifier′ ∈ $rClass′ ∪ getSuperclassifiers($rClass′) do
        allRClassifierNames′ ⟵ allRClassifierNames′ ∪ getName($rClassifier′);
    rClassName ⟵ getName(getClass($reference));
    if $rClassName ∉ $allRClassifierNames′ then
        ⟵ "not referencing class $rClassName any more";
    else ⟵ ∅;
end
```

Finally, Algorithm 3 (called at Line 24 of Algorithm 1) implements the conclusions from Section 2.3.4 effectively with conditional statements. This is possible as the statements operate on the upper and lower bounds of the multiplicities. If the lower bound is changed and set to zero (Line 2), this introduces optionality (addressing Conclusion 10). If an original upper bound with the original value of one was changed (Line 5), it may introduce the necessity to iterate over a set of elements instead of accessing one single element (addressing Conclusions 11 and 12). In case the upper bound was lowered (Line 7), a warning is raised, as this may break navigating expressions that access an element above the new upper bound. Finally, the algorithm allows for removing the optionality of a multiplicity (covering Conclusion 9).

Note that the algorithm does not terminate in case of an error, e.g., a missing relationship, but continues to process the model change. It collects all errors and warnings and returns this together with a logical result that signifies the navigation compatibility of the model change. Using the keys from the `errors` and `warnings` hash tables the user or a modeling tool can identify the respective model element and, e.g., set an focus on it.

2.5 Evaluation

In this section we will first present results from analyzing model changes of a MDD project. We will then discuss about the runtime complexity of the algorithm and show performance evaluations from real world meta-models. Finally, we present some lessons learned and conclude with experiences from applying our algorithm in an industrial setting.

Algorithm 3: Multiplicity Check

```
   Input: reference, reference′
   Output: warnings
1  begin
2      if getLower($reference) $\neq 0 \wedge$ getLower($reference′) $= 0$ then
3          warnings $\longleftarrow$ “optionality introduced”;
4      else warnings $\longleftarrow \emptyset$;
5      if getUpper($reference) $= 1 \wedge$ getUpper($reference′) $\neq 1$ then
6          warnings $\longleftarrow$ $warnings $\cup$ “iteration necessary”;
7      if getUpper($reference) $>$ getUpper($reference′) then
8          warnings $\longleftarrow$ $warnings $\cup$ “upper bound lowered”;
9      $\longleftarrow$ $warnings;
10 end
```

2.5.1 Analyzing MDD Projects

With our ready to use algorithm we conducted navigation compatibility analysis of model changes within model-driven projects. In Table 2.2 some results from the Model-Aware Service Environment [82, 87, 86] (MORSE) project are summarized. We can see that the constructive model that is used by the project for code generation was modified several times (only the first results from the changes are displayed). While changes to the model from Δ_5 on produced navigation compatible models (`n(Errors)` $= 0$) the previous changes introduced navigation incompatibilities. This is because the model was refactored in the beginning of the project and thus was not stable yet.

Sometimes model elements simply have been renamed. Yet, for such renames the algorithm produces errors as a model transformation that hardcodes names of model elements such as classes or role names need to be aligned accordingly. Nonetheless, it would be desirable if the algorithm could consider renames and instead of raising an error issues a warning for the renamed model element. This is particularly useful if the MDD developer is aware of the rename and/or it has been considered in transformations that process conforming model instances.

By extending the algorithm accordingly and by specifying a set of renames for the model changes errors of the model change Δ_1 could be reduced and for Δ_3 even eliminated [6]. That is, we can say that the model change Δ_3 resulted in a navigation compatible model when considering renames.

All checks completed within a fraction of a second and are thus satisfactory for our purposes. Note that without considering renames for Δ_1 the algorithm terminates rapidly as correspond-

[6]The results are found in the right, *renames* labeled columns.

ing classes or references are not found and a further check simply does not to take place (cf. Line 8 and 17 of Algorithm 1).

Table 2.2: Model Changes of the MORSE Project

Δ	**Compatible**		**RunTime [ms]**		**n(Warnings)**		**n(Errors)**	
		renames		renames		renames		renames
Δ_1	false	false	**56**	267	0	**11**	**13**	**2**
Δ_2	false	false	406	327	0	0	9	9
Δ_3	false	**true**	228	215	0	**1**	**19**	**0**
Δ_4	false	false	164	162	0	0	20	20
Δ_5	true	true	159	170	0	0	0	0
Δ_6	true	true	193	221	0	0	0	0
Δ_7	true	true	159	177	0	0	0	0
Δ_8	true	true	161	163	0	0	0	0
Δ_9	true	true	146	155	0	0	0	0
Δ_{10}	true	true	140	143	0	0	0	0
Δ_{11}	true	true	146	137	0	0	0	0
Δ_{12}	true	true	133	126	0	0	0	0
Δ_{13}	true	true	119	109	0	0	0	0
Δ_{14}	true	true	114	121	0	0	0	0
Δ_{15}	true	true	113	109	0	0	0	0

2.5.2 Runtime Complexity

The presented Algorithm 1 has a run time complexity of

$$\mathcal{O}(c'(c+r+r')) \tag{2.1}$$

with c = number of classes in the original model and c' in the changed model and r = the number of total references in the original model and r' in the changed model. For a model change with small changes we can conclude:

$$c \approx c' \wedge r \approx r' \implies \mathcal{O}(c^2 + cr) \tag{2.2}$$

If we search for a large navigation compatible model m' using a small model m we can expect the following run time complexity:

$$c \ll c' \wedge r \ll r' \implies \mathcal{O}(c'r') \tag{2.3}$$

As we will show next, for the MDD scenarios generally, the runtime is far satisfactory.

2.5.3 Performance Evaluation

For evaluating the performance of the algorithm we have conducted tests with 286 real world meta-models, some – such as shown in Figure 2.1a – originating from industrial companies, others have been retrieved from open source projects as well as the Atlantic Meta-Model Zoo [12]. Table 2.3 reflects the metrics of these models such as the number of classes (`n(Classes)`) and number of references (`n(References)`). These are summarized together with the results from our evaluation in Table 2.4.

From the collected metrics we can assume typical MDD meta-models to have certain characteristics: First, the number of references in meta-models is in the same dimension as the number of classes. For such meta-models we can therefore – by refining Equations 2.1 and 2.2 – state a run time complexity of

$$\mathcal{O}(c'c) \approx \mathcal{O}(c^2) \mid c \approx c' \tag{2.4}$$

Second, we can assume typical MDD meta-models to have a size of less then twenty classes. Indeed, half of the models contain not more than twelve classes (see also Figures 2.6b and 2.6c).

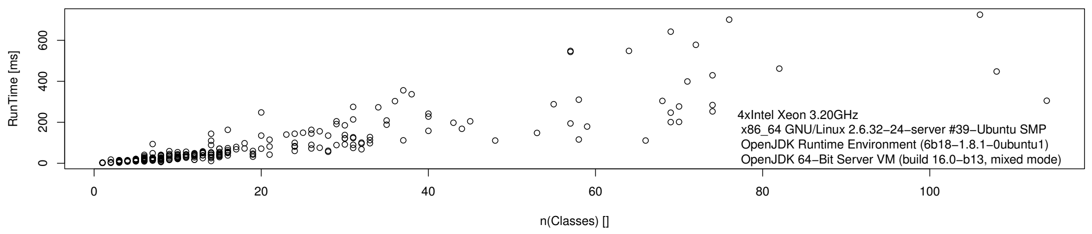
Table 2.3: Metrics of Industrial and Open Source Meta-Models

	Min.	**1st Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max**	
	3.00	7.00	8.50	11.43	11.75	29.00	n(Classes)
Austrian Telecommunication Provider	0.000	0.000	0.500	3.714	5.750	16.000	n(Generalizations)
$n(Models) = 14$	2.00	7.00	13.50	14.79	20.50	37.00	n(References)
	5.00	7.25	12.00	15.00	21.50	32.00	n(Classes)
European MVNO	0.0	0.5	4.5	7.6	10.5	26.0	n(Generalizations)
$n(Models) = 10$	6.0	8.0	13.0	15.6	18.5	42.0	n(References)
	1.00	3.75	8.50	18.93	27.25	74.00	n(Classes)
EMF Projects [162]	0.00	0.00	6.50	16.95	21.25	83.00	n(Generalizations)
$n(Models) = 44$	0.00	3.75	9.50	19.89	29.50	125.00	n(References)
	1.00	7.00	14.00	21.02	27.25	114.00	n(Classes)
Atlantic Meta-Model Zoo [12]	0.00	3.00	9.00	15.61	20.50	96.00	n(Generalizations)
$n(Models) = 188$	0.00	7.00	14.00	27.04	29.00	231.00	n(References)
	30	30	30	30	30	30	n(Classes)
MORSE [82]	26	26	26	26	26	26	n(Generalizations)
$n(Models) = 1$	20	20	20	20	20	20	n(References)
	3.000	6.000	7.000	9.273	10.000	27.000	n(Classes)
VbMF [173]	0.000	0.000	0.000	2.636	4.500	14.000	n(Generalizations)
$n(Models) = 11$	0.00	5.50	8.00	10.18	12.50	33.00	n(References)

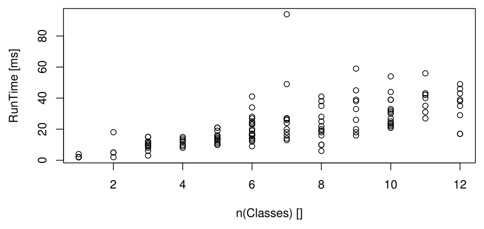
We conducted the checks on the models without a change (i.e., $m = m'$). This way the *entire* model with all its classes and relationships is checked by the algorithm. This equals the worst case for the runtime and thus the execution is maximized. The results from evaluating the runtime of the algorithm with the meta-models are displayed in Figure 2.6a. Figures 2.6b and 2.6c display half of the population using the median $n(Classes) \leq 12$. Figures 2.6a and 2.6b show the runtime as a function of the number of classes. Similarly, the abscissa in Figure 2.6c uses the total number of references in the model. All test results (summarized in Table 2.4) show a runtime that is less than a second. The average execution time of all models is $87.1ms$. Therefore the presented algorithm proves far satisfactory for MDD scenarios.

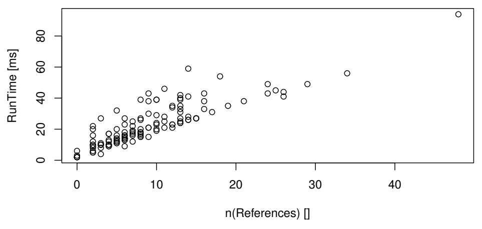
Table 2.4: Summary of Meta-Models Metrics

Min.	1st Qu.	Median	Mean	3rd Qu.	Max	
1.0	6.0	12.0	19.5	26.0	114.0	n(Classes) []
0.00	2.00	7.50	14.42	16.25	96.00	n(Generalizations) []
0.00	6.00	13.00	24.08	27.25	231.00	n(References) []
0.0000	0.2857	0.6000	0.5537	0.8632	1.7140	n(Generalizations) / n(Classes) []
0.0000	0.7411	1.1500	1.2450	1.5770	6.8570	n(References) / n(Classes) []
2.0	18.0	39.5	87.1	98.5	725.0	RunTime [ms]



(a) Runtime of the Algorithm using 286 Real World Models





(b) Runtime as a Function of the Number of Classes in a Model using the Median $n(Classes) \leq 12$

(c) Runtime as a Function of the Number of References in a Model using the Median $n(Classes) \leq 12$

Figure 2.6: Performance Evaluation of the Navigation Compatibility Check

2.5.4 Lessons Learned

When studying the results and looking at the errors identified by the algorithm, we found that they sometimes result from simple renames of model elements. Such renames in a model need to be aligned with corresponding *navigating expressions* (see Section 2.1), e.g., OCL expressions. Supposed a MDD developer uses an integrated development environment (IDE) that supports the rename of model elements by aligning navigating expressions as used in transformation rules, we would like the algorithm to consider the renames for the navigation compatibility check as well. For supporting this scenario we have conducted a slight adaptation: (possibly renamed) class and role names are resolved after Line 5 and 13 in Algorithm 1. Another finding was the removal of abstract superclasses. It is reasonable that such modifications do not result in errors (Line 7) in case of missing abstract superclasses, as such classes cannot be instantiated. Of course if an abstract superclass is referenced, still an error is issued (Line 15).

Statements can be given if changes are necessary to *navigating expressions* in model transformations after model changes using our algorithm. Yet we do not answer the question if any changes in model transformations become necessary after a model has been changed. Besides navigability issues there can be other model changes that require adjustments in model transformations and thus introduce other incompatibilities. Thus, our presented work focuses and is limited to the *navigation* compatibility of model changes.

2.5.5 Navigation Compatible Changes Applied in an Industrial Setting

Our algorithm has been applied, besides in the MORSE project, by an European MVNO during the MDD and evolution. In this project, changes to meta-models such as shown in Figure 2.1a have been evaluated continuously regarding their navigation compatibility. This was done for ensuring the downwards-compatibility of evolved meta-model versions.

Of course, restricting the evolution of meta-models to navigation compatible changes only makes limited sense. Herrmannsdörfer [81] describes the various drawbacks of such a downwards-compatible evolution: On the one hand, the way in which a meta-model may evolve is heavily restricted. On the other hand, the preservation of old constructs clutters and complicates a meta-model. Despite these limitations, with a navigation compatible change, the meta-model is adapted in such a way, that old models still conform to the new meta-model. Thus, they do not have to be migrated. Also transformation templates do not need to be adapted.

For minimizing the effort of co-evolution, the company applied the following strategy:

- Whenever possible meta-model changes should be navigation compatible, MDD developers are free to undertake such changes frequently.
- Changes that break navigation compatibility, however, require a formal agreement of the developers as this involves the co-evolution of other artifacts such as models and transformation templates.

As a result, the evolution of meta-models and related artifacts can be characterized by either lines of navigation compatibility or moments of navigation incompatible changes.

If we compare this kind of evolution to the evolution of software libraries we find similarities. A new version of the software might introduce major changes, some of them may break the downwards-compatibility. Nonetheless, updates may come with downward-compatible, non-breaking changes within a certain version. An API of a software generally is not subject to breaking changes. This way a developer who works with the software can assume updates of at least the same software version to conform to the former API. When upgrading to a new version the developer should be aware of the fact that this might break his code and that tests need to be performed.

From our experience, we believe that MDD can profit from the experience and findings of software evolution. Particularly, we found the implementation of navigation compatible changes helpful for the evolution and maintenance of MDD projects. Meta-models may be managed in different, navigation-compatible lines of versions. A set of MDD artifacts and systems is then known to work with a certain navigation-compatible model. By benefiting from navigation compatibility between small changes, this strategy does not exclude complex evolution to take place.

2.6 Related Work

In this section we mention on work in the fields of model comparison and model co-evolution and relate to our work.

Model Comparison

When we check if a model is navigation compatible against another model we *compare* these models that can be two versions of a model representing a model change. The literature on model comparison includes a multitude of related work, e.g., on model *differences* (e.g., [32, 6, 136]) and model *merging* (e.g., [6, 28, 102]). The former category is relevant to our work. For example we need to check if classes or role names have been removed. A systematic overview of model comparison approaches is given by Selonen [145]. He presents qualities of model comparison techniques for evaluation and identifies common usage scenarios. Also, there is industrial realization: the IBM Rational Software Architect has some support for the comparison and merging of UML models [101]. For example it is possible to visualize differences between model versions in a local history. Finally, the Eclipse Foundation also hosts an EMF Compare [164] project that aims at providing a framework for model comparison. For this the comparison process is divided in a matching and differencing phase. In contrast to this two-step comparison our algorithm processes two models and saves errors and warnings in one run. Moreover, to the best of our knowledge, the specific problem of *navigation compatibility* as presented in this chapter has not been focused at.

Ohst et al. [136] specifically focus on UML models and the visualization of differences. Besides the structure of models the work also considers the layout of UML diagrams. As explained in Section 2.1 our approach also operates on the meta-model $M2$ level and can work with instances of, e.g., UML or Ecore [162] meta-meta-models. Few details are given about the implementation. While the computation relies on a repository that stores the models and model elements for comparison, our algorithm does not require the use of a repository. Instead two serialized models can be processed by our algorithm. For this currently we support two different $M3$ modeling languages.

Alanen and Porres [6] formally discuss the difference and union of models. Their implementations are meta-model independent but rely on a version control system (VCS). While their work is generic to the problems of model differences and unions our work specifically focuses on and is suited for the navigation compatibility of models.

Egyed [58, 59] presents rules and algorithms for abstracting "low-level" models to more abstract "higher-level" models. Although, his work focuses on representing models at different abstraction levels to humans, e.g., to different stakeholders of a software system, we found this work quite suited for our purpose of model comparison. However, the transitive abstraction presented does not consider the use case of navigation compatibility. Therefore, we had to discuss the following points that arose within the context of MDD and that have not yet been covered by his work: first, *navigability* between classes and of *relationships* could in some cases be simplified for navigation compatibility, in others not. Second, the notion of *roles* is important in regard to relationships. Finally, *multiplicities* between classes should be considered as well as they are usually handled differently during the code generation step.

Model Co-Evolution

As the evolution of meta-models, poses a thread to the applicability of MDD, Gruschko et al. [73] discuss the problem of model *erosion* that is caused by changes in corresponding meta-models. They classify changes such as renames, deletion, addition, movement as well as changes of association and type and propose a generic, semi-automated approach to the model migration.

A realization was carried out by Cicchetti et al. [42]. First, a (calculated) difference model captures the meta-model evolution. Finally, model transformations are produced for co-evolving conforming models. Besides atomic changes such as described in [73], and in order to deal with realistic scenarios, they particularly focus on combination of changes that they categorize into parallel independent and dependent changes.

Herrmannsdörfer et al. [81] propose COPE that realizes so-called *coupled transactions*, i.e., the coupled evolution of meta-models and models. In addition to *custom* coupled transactions that can be defined by a developer for complex migrations and that are specific to a meta-model, a *reusable* coupled transaction can be used for recurring coupled transformations across meta-models. For tool support, a non-invasive integration of COPE into the Eclipse Modeling Framework [162] (EMF) meta-model editor has been realized.

Wachsmuth [177] defined several semantics- and instance-preservation properties in terms of meta-model relations. For supporting the automatic meta-model evolution in a stepwise adaptation, the authors also present a set of transformations.

All of these works focus on the meta-model/model co-evolution. Thus changes that occur to meta-models are studied regarding their impact on conforming models. One of the results is the level of automatization that can be realized for co-evolving models. In contrast to meta-model/model co-evolution, our work concerns meta-model/transformation template co-evolution. Particularly we focus on navigating expressions and non-breaking changes to meta-models.

Levendovszky et al. [114] present an approach to semi-automated evolution of model transformation. With their work they target the problem of co-evolving transformations for meta-models. Thus, for a given model transformation $t : MM_{src} \rightarrow MM_{dst}$ a model transformation $t' : MM'_{src} \rightarrow MM'_{dst}$ is wanted for the evolved meta-models. The authors present categories of transformation operations that are fully or partially automated or fully semantic. In their work, *subtyping* is only mentioned. Thus, details are missing for generalization and realization relationships such as we have presented with Algorithm 2. Also changes to multiplicities (cf. Figure 2.5 and Algorithm 3) or the navigability (cf. Figure 2.4) of relationships are not discussed. In this chapter, in contrast, we have conducted a focused study on the topic of navigation compatibility that we believe contributes to better support the (semi-automated) evolution of meta-model co-evolution, e.g., model transformations.

2.7 Summary

Starting and inspired from the idea of transitive reasoning and related work we analyzed the impact on the navigability of different model modifications. In our studies we identified a set of conclusions that are supported by our algorithm that we presented in Section 2.4. This algorithm determines the *navigation compatibility* of a model change, e.g., between two versions of a model and gives detailed *feedback* in form of warnings and errors. This information in turn can be used by a developer for interactive development. Using our algorithm, software developers can run checks prior to the deployment of software systems in a new version. Such checks are particularly meaningful during the modeling phase, i.e., at design time and after a feedback loop from the runtime.

In Chapter 5, e.g., we will make use of several meta-models such as displayed in Figure 5.5 that are subject to change. When these meta-models change, our algorithm – exposed as a service within MORSE that we will present in the next chapter – helps developers to avoid or identify navigation incompatibilities. Figure 6.3 in Chapter 6 displays a similar meta-model that changed during a case study (cf. 6.2). In this case study, our Algorithm 1 that we presented for determining the navigation compatibility of a model change eased the evolution of the system and the monitoring.

Future Work

The algorithm does not have to operate on a model change necessarily. By specifying a model m for Algorithm 1 we can search for some navigation compatible models m'. Thus, we plan to extend and apply our algorithm for search scenarios in future.

CHAPTER 3

Model-Aware Service Environment

Rien n’est plus dangereux qu’une idée, quand on n’a qu’une idée. [1]
— ÉMILE CHARTIER

Contents

[1] *Nothing is more dangerous than an idea, when it’s the only one we have.*

Usually in model-driven engineering (MDE) model management and use takes place at design time. Yet, currently there is no collaborative development environment (CDE) dedicated to the domain of MDE, i.e., for managing MDE projects. In various model repositories, identification and retrieval of models, model elements, and other MDE artifacts – if supported by the repository at all – in a distributed environment is sometimes not evident. This is mainly due to problems arising from the versioning of these artifacts. Our approach to solve this issue is a Model-Aware Service Environment [82, 87, 86] (MORSE). It consists of a service-oriented model repository that manages MDE projects and artifacts, and model-aware services (explained in the next chapter) that interact with the repository for performing reflective queries on the models stored in the repository. Thus, MORSE supports the dynamic, reflective look-up of models in service-oriented systems and enables collaborative development in the context of MDE.

3.1 Introduction

Many model-driven development (MDD) approaches for service-oriented architectures (SOAs) have been proposed (e.g., [153, 20, 123]). These approaches are model-driven in the sense that the SOA is specified using models and large parts or the whole source code of the SOA (including for example Web service code, Web Services Description Language [40] (WSDL) files, policy code, business object implementations, and so on) is generated from those models.

While the model-driven SOA approach is highly useful in many cases, it has its limitations for scenarios that require information from the models at runtime because generation currently happens only at design time. Hence, all information that is needed at runtime from the models must be foreseen by the developers and must be statically generated into the source code. Many SOAs require "reflective" model information for monitoring, auditing, reporting, and business intelligence purposes. The requirements for this kind of reflection on model information can change quickly and are hard to foresee. However, regenerating and redeploying major parts of the SOA source code, because a certain model element's information is not exposed, is often not feasible in large distributed architectures. In addition, developing new reflection functions for each single model element is costly.

Finally, model information that is generated into or attached to a model-driven service is only up-to-date at its generation time. This is problematic when the service needs to reflect on information that was supplied after its generation and deployment. This is particularly true in a distributed and persistently evolving environment. Statically generated services need to keep up with changes such as additional model relations, e.g., new annotation models or a new version of the model of a service.

Our approach to solve these issues, i.e., facilitate services to dynamically reflect on models, is to create model-aware systems for the SOA and support these with a service-based model repository. We call such a SOA a *Model-Aware Service Environment [82, 87, 86] (*MORSE*)*. During the MDD process, each model of the SOA is placed in the model repository. Each model and model element gets a Universally Unique Identifier [113, 98] (UUID) assigned, with which the model or model element can be uniquely identified. The UUIDs are generated into the source code of the model-aware services (and components). Hence, they are *model-aware* in the sense that they can retrieve the models from which they have been generated from the repository at runtime using a service. In the same way, other components such as monitoring, auditing, reporting, and business intelligence components, or MDD tools such as a model-driven generator, can retrieve these models. The repository service interface is a generic interface, and the UUIDs are generically added to generated code. Hence, no changes of the generated SOA are necessary in order to use a model element at runtime that has not been used before. Also the model-aware services can access evolved models and model relationships that occurred after generation time of the model-aware service.

3.2 Approach

For facilitating services and MDD developers to dynamically work with models in a SOA, we propose the Model-Aware Service Environment [82, 87, 86] (MORSE). MORSE consists of a model repository and model-aware services that interact with the model repository using service-oriented interfaces (i.e., the MORSE services; see Section 3.5).

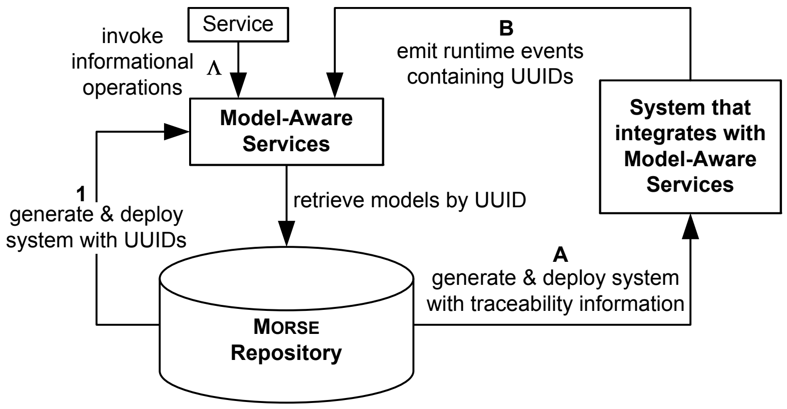

Service
invoke informational operations
Λ
B
emit runtime events containing UUIDs
Model-Aware Services
System that integrates with Model-Aware Services
1
generate & deploy system with UUIDs
retrieve models by UUID
A
generate & deploy system with traceability information
MORSE Repository

Figure 3.1: Overview of the Model-Aware Service Environment

Figure 3.1 (that will be further explained and used in the next chapter) gives an overview of the MORSE. From the repository model-aware services can be generated that interact with the information retrieval (IR) interface. Also services and processes with traceability information that emit events to model-aware services can be generated (presented in the following chapter).

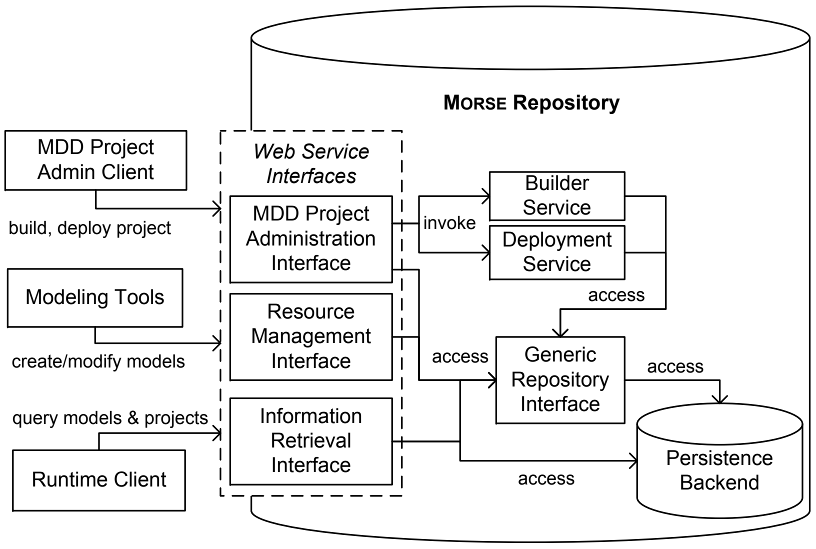


Figure 3.2: Architecture of the MORSE Repository

Figure 3.2 gives an high-level overview of the model repository architecture. Different Web service interfaces allow for the administration and resource management of MDD projects and artifacts and offer IR functionality (see end of the following section) to model-aware services. The models are created by human modelers using modeling tools. Using admin clients, MDD projects in the MORSE repository can be created. Import clients allow the modelers to add models, model elements, and model relationships, or to provide them in a new version. The MORSE builder service can create the model-aware services. Also it can weave UUIDs [2] of MORSE objects into generated code. A deployment service is used for deploying resulting services and processes on runtime engines (see Section 4.3).

Because all MDD projects and artifacts are managed in model repositories (see next section), model-aware services can query these for any information on themselves and other model-driven components. Although models and model relations may evolve over time, using UUIDs, it is possible to retrieve a specific version of a MORSE object (this will be explained in more detail in Section 3.4.2). Derived versions, e.g., new versions of the model that were created after deployment time, can easily be identified, permitting a model-aware service to, e.g., retrieve and work with the latest version of a model.

[2]Universally Unique Identifier are used in MORSE to uniquely identify models and model elements across distributed components, such as the model repository, model-aware services, and other components using the MORSE services such as monitors.

MORSE can also be beneficial for MDD tools, e.g., in a distributed CDE [27] while fostering service-orientation to support the MDD design time tooling. In this case, not the model-aware services or components monitoring them would retrieve and change the models, but the MDD tools such as a model-driven generator.

3.3 A Collaborative Development Environment for MDE Projects

CDEs offer virtual spaces to stakeholders involved in distributed development. Booch and Brown [27] enlist coordination, collaboration, and community building as common features of CDEs and mention *centralized information management* as essential to the coordination aspect. Applied to MDE – as an emerging software engineering discipline – management of artifacts such as models, transformations, and templates is required. In order to support MDE with a CDE, thus, an appropriate management of these MDE artifacts is essential. There are a couple of CDEs dedicated to the domain of software engineering (SE) such as Codehaus [44], CollabNet [46], GForge [69], or Launchpad [35]. However, they do not yet specifically target MDE. Notwithstanding the lack of CDEs for MDE, *model repositories* are used for MDE artifact management during MDD (cf. [64]).

In our concept, MDE artifacts and their management in repositories is essential for the model-aware systems. That is, model-aware systems may interact at design- or runtime with a repository, e.g., for model look-up. Some resulting benefits are

- For collaborative development, MDE projects and artifacts can be managed with version control.
- MDE artifacts can be searched (e.g., for reuse scenarios) and explored.
- MDE artifacts can be retrieved at runtime for dynamic, model-based execution (cf. [78]).

3.3.1 Challenges on Deploying Model Repositories

Despite these benefits, repositories that manage MDE projects and their artifacts, particularly models, are rarely employed as central components of model-driven systems and offer only limited basic functionality. This is because design and development of such repositories is complex:

- They use various technologies and need to be adapted to support emerging technologies and requirements.
- They introduce dependencies to other MDE components, and proper system integration becomes crucial for regular operation. An adequate repository may be connected with

components for monitoring, notification, traceability, and so on. It is usually difficult to modify or even replace a once deployed repository.

- Last but not least, interfaces and implementation of model repositories consist of recurring code, such as code for storage and retrieval, that – if not supported by MDD – becomes cumbersome to maintain.

3.3.2 A Model-Driven Approach

We address these issues by proposing a model-driven approach for developing repositories as key components of a MORSE. That is, MORSE not only supports MDE, but is also itself built using MDE techniques.

MORSE repositories allow for describing dependencies of MDE projects and artifacts. For this purpose we make MDE artifacts uniquely identifiable in a distributed setting using UUIDs. Finally, MORSE allows for the continuous integration (CI) (cf. [57]) of repositories with various MDE components because the generation, build, and test of MORSE repositories have been fully automated.

Our approach can be used to modify a CDE to support MDE. However, this would cover only design time support for MDE tasks. Our approach can also be used for supporting the runtime and other tasks of the MDE process.

While MORSE repositories support MDE projects, MORSE itself has been developed following the MDE approach. Therefore we start by illustrating a conceptual model [3] – that can be customized if necessary – from which MORSE repositories are generated [4].

3.3.3 Conceptual Model of MDE Projects and Artifacts

The model repository is the main component of MORSE [87] and has been designed with the goal to abstract from specific technologies. Thus, while concepts are taken from, e.g., the UML and also version control systems (VCSs) (cf. [115, 121, 119, 159, 55]), MORSE is particularly agnostic to specific modeling frameworks or technologies.

[3]We designed our model using the Eclipse Modeling Framework [162] (EMF) [34]. The resulting Ecore model that is serialized in the XML Metadata Interchange [135, 95] (XMI) format is based on Essential Meta Object Facility [131] (EMOF), a Unified Modeling Language [134] (UML) [134] based meta-model that is part of the Meta-Object Facility [131] (MOF) [131] of the Open Management Group (OMG).

[4]From this model we derive interfaces and implementations for IR and management of MDE projects and artifacts. We integrate various technologies in model-to-code templates. IR is eased by exploiting the relationships between MDE artifacts. MORSE repositories provide versioning capabilities not only to the artifacts, but also their relationships. For safeguarding the functionality of MORSE repositories we realized a test-driven approach by deriving automatic tests and integrate MORSE with a system for CI.

The MORSE repository manages objects (`MObject`) such as projects (`MProject`) and artifacts (`MArtifact`) as shown in Figure 3.3 and 3.4a. Additional MORSE object types (explained below) are shown in Figure 3.4b. All `MObjects` are identifiable by `uuids` and can be associated with Dublin Core [108] (DC) metadata such as `title`, `creator`, or `date`. Note that a UUID uniquely identifies a particular MORSE object. We will cover the topic of versioning in the next section. By navigating across the `original` or `modified` relations, previous and derived versions of a certain version can be identified.

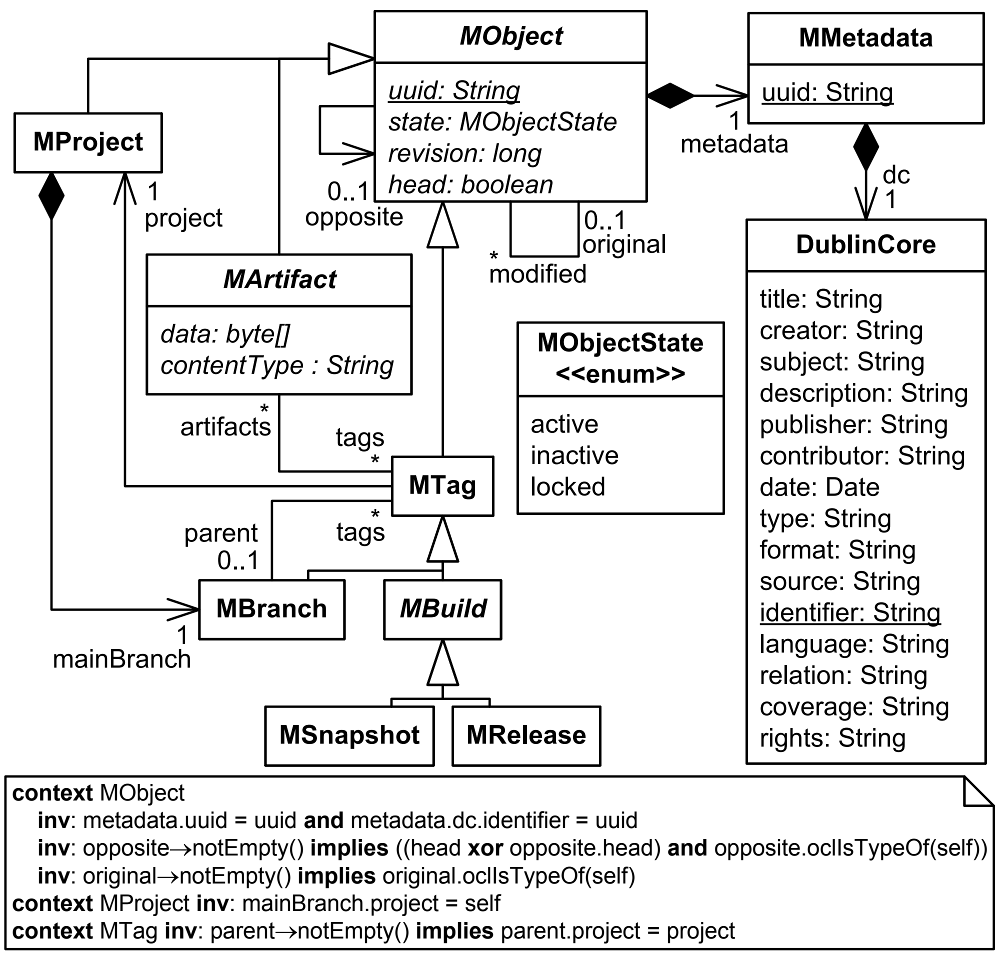
Artifacts are used to manage models and model elements (for details see below), model transformations, and MDD workflows. Besides the versioning of these, the repository supports branching (`MBranch`) and tagging (`MTag`) of projects. Note that artifacts can be shared by multiple projects as they can be associated by different tags and branches. They can be changed independently and merged later on.

Typical MDD projects consist of models, transformations, and workflows. An example of a MDD development framework that works with these artifacts is openArchitectureWare [138] (oAW). We have adopted these artifact types and support them in MORSE as shown in Figure 3.4a.

Besides the general management of MDD artifacts, MORSE particularly realizes support for models. Models typically contain model elements and have relationships to other models. By capturing and keeping track of these, MORSE facilitates reflection on models, model elements, and model relations. Figure 3.5 illustrates `MModel`, `MModelElement`, and `MModelRelation` classes that represent these concepts. All these classes derive from `MObject` and are identifiable and versionable as such. Moreover, `MModels` are `MArtifacts` and can use the `data` attribute to save a serialized form of a model. Examples of different relations are instance-of (`MInstanceRelation`), inheritance (`MInheritanceRelation`), and annotation (`MAnnotationRelation`) relations as shown in Figure 3.4b. A model relation (`MModelRelation`) has a source (`src`) and destination (`dest`) model, e.g., an annotated model is the destination model of a `MAnnotationRelation` and the annotation model depicts the corresponding source model. Besides referring to the models, a model relation may also specify actual model elements (`srcElement` and `destElement`).

While it would be possible to further specify details, e.g., of model elements, the presented concepts are sufficient for our purposes, i.e., to make models and model elements identifiable and to capture dependencies between different models as introduced though their relations. The models that are stored and versionised within `MModels` can be retrieved in a serialized form and can further be processed by technology-specific tools, e.g., for introspection, model transformation, or model validation.

For the presented classes and concepts, the model repository exposes different services as



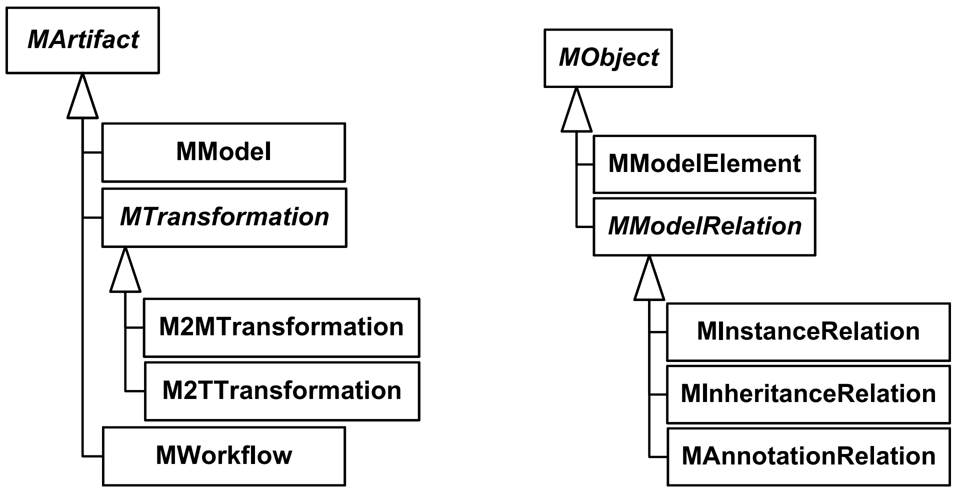
```
context MObject
   inv: metadata.uuid = uuid and metadata.dc.identifier = uuid
   inv: opposite→notEmpty() implies ((head xor opposite.head) and opposite.oclIsTypeOf(self))
   inv: original→notEmpty() implies original.oclIsTypeOf(self)
context MProject inv: mainBranch.project = self
context MTag inv: parent→notEmpty() implies parent.project = project
```

Figure 3.3: MORSE Objects and Projects

indicated in Figure 3.2. Besides an administrative and resource management interface, the repository particularly offers an IR interface to model-aware services.

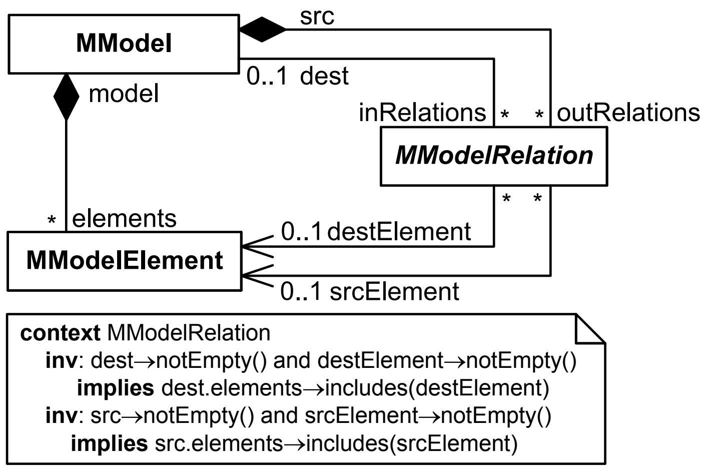
3.4 Model Versioning

Common VCSs such as Concurrent Versions System [55] (CVS), Subversion [159] (SVN), Bazaar [119], Git [115], or Mercurial [121] are not suitable for model versioning. This is because they are only able to store serialized forms of models and problems may arise when concurrent changes occur. That is, these changes may not affect each other but still conflicts may occur due to the serialization of the model. Besides the versioning aspect, also the search for models in such repositories is unfavorable. Therefore we aim for a repository that is particularly tailored



(a) MDD Artifacts (b) additional MORSE Objects

Figure 3.4: MDD Artifacts and additional MORSE Objects



```
context MModelRelation
  inv: dest→notEmpty() and destElement→notEmpty()
     implies dest.elements→includes(destElement)
  inv: src→notEmpty() and srcElement→notEmpty()
     implies src.elements→includes(srcElement)
```

Figure 3.5: Model Element and Model Relations

for models and the management of model versions.

3.4.1 MORSE Repository

A model contains model elements such as *classes*, *attributes*, and *references* (cf. Ecore [162] and EMOF [131] *classes* and *properties*). In the MORSE repository each instance [5] of a class

[5]M2, meta-model class (cf. to the OMG [131] definition)

corresponds to a table in a relational database [6]. An instance thereof is stored as a row in such a table. A respective record thus holds the values for the respective attributes and also the references in the form of foreign keys and is the smallest unit of versioning (cf. [137, p.4]) (UV) in the MORSE repository. If the value of an attribute changes, the record is updated. Similarly, the records are updated in case references are set, deleted, or changed. As a value the UUIDs of the opposite model class instance(s) are used.

3.4.2 Transparent UUID-Based Model Versioning

In a MORSE, models (i.e., meta-models and conforming models (cf. [23])) are stored in and accessible from model repositories. This repository needs to support the versioning of models, as different versions of models can be used by different services. At runtime the services that dynamically interact with the repository need to retrieve specific model versions. In contrast, modeling tools and end users such as system stakeholders typically expect a *transparent* versioning. That is, they can reuse an identifier of a previously updated model for obtaining the current model in its latest version.

Problems arise however when UUIDs are used as identifiers as they are required to be unique "across both space and time, with respect to the space of all UUIDs" [113]. Thus, two versions of a model or model element cannot be identifiable by the same UUID [7]. If this is not respected in versioning, UUIDs get duplicated and when this happens they *degenerate* into ordinary identifiers (IDs) as they will not be unique any more.

In software configuration management [88, 92] (SCM), usually an additional identifier for specifying a particular version is used. In contrast with UUID-based identification no composite identifiers are possible. While this restriction may be regarded as a limitation of UUIDs, this is on purpose and comes with the advantage of a unique identification. Thus, no additional information or identifiers are required but a single UUID suffices to specifically and uniquely identify an object.

As we aim to adopt models at runtime and need to relate to models and model elements easily, the idea to uniquely identify a model or model element in a distributed environment is important to our approach. Thus, UUIDs appear not only appropriate but also appealing due to their simplicity: a UUID has a size of 128 bits and is represented in its canonical form as a 36 character string, consisting of 32 hexadecimal digits and four hyphens as depicted in Table 3.1.

A UUID can be generated autonomously without the need for a central authority and can

[6]The MORSE repository is realized with Teneo [167] that utilizes the EMF. For the persistence we chose EclipseLink [163] as a Java Persistence API [54] (JPA) implementation and a relational database management system (RDBMS) such as PostgreSQL [140] or Apache Derby [156].

[7]In contrast, a model may be identifiable by multiple UUIDs.

Table 3.1: Canonical and URN Representations of a UUID

Canonical Form of a UUID	1787475b-5a32-48b9-9b8a-6d813bfc6e51
URN Representation of a UUID	urn:uuid:1787475b-5a32-48b9-9b8a-6d813bfc6e51

easily be exchanged in form of a Uniform Resource Name [77, 120] (URN) (cf. [113]; see also Table 3.1). Yet, in order to apply them for model management and versioning a problem had to be solved: how to realize a transparent UUID-based versioning of models. In the following we propose a solution to this problem that we realized for the versioning of models in the MORSE repository.

With our transparent UUID-based model versioning technique we propose to store – in addition to version-specific models – a copy of the latest version in the form of a version-independent model (head). All of these models and its model elements are identifiable by UUIDs. Hence, for the identification of models we distinguish between version-independent UUIDs (VIIDs) and version-specific UUIDs (VSIDs). Thus, a VIID/VSID identifies models or model elements of the head/a certain revision. Note that by distinguishing between version-specific objects and the head we respect the uniqueness property of UUIDs as in our approach a UUID particularly identifies either a version-independent model (element) or a model (element) from a specific version. Across all models, model elements, and all their versions any UUID is just used once and thus is unique. There are further characteristics: version-specific UUIDs identify objects that are not subject to change. In contrast, an object identified by a version-independent UUID is. Thus, if a certain version of an object is required, e.g., by a runtime service, it is best identified by a VSID. In contrast, for working with the up-to-date version VIIDs can be used, e.g., by modeling tools.

Figure 3.6 depicts our solution for the transparent UUID-based model versioning. On the left hand side, the version-independent model (`Head`) is displayed in four different revisions. On the right hand side, the version-specific objects are displayed. Note that in the MORSE repository (see Figure 3.3) the `head` property of an `MObject` indicates if the object is version-independent or -specific. Also note that a corresponding `opposite` object is navigable. Thus, a version-independent object references the latest version-specific object and a version-specific object holds a reference to the version-independent object (if existent).

Typically a model change introduces new, changes existing, or eliminates model elements. That is, not the entire model is updated in an evolution step but only parts of it change. Our model versioning approach respects this and operates on small changes in a space saving way.

In Revision 1 two model elements `A` and `B` exist. The first change introduces a reference from the former to the latter. In MORSE the reference is stored as part of `A`. Therefore, `A` is

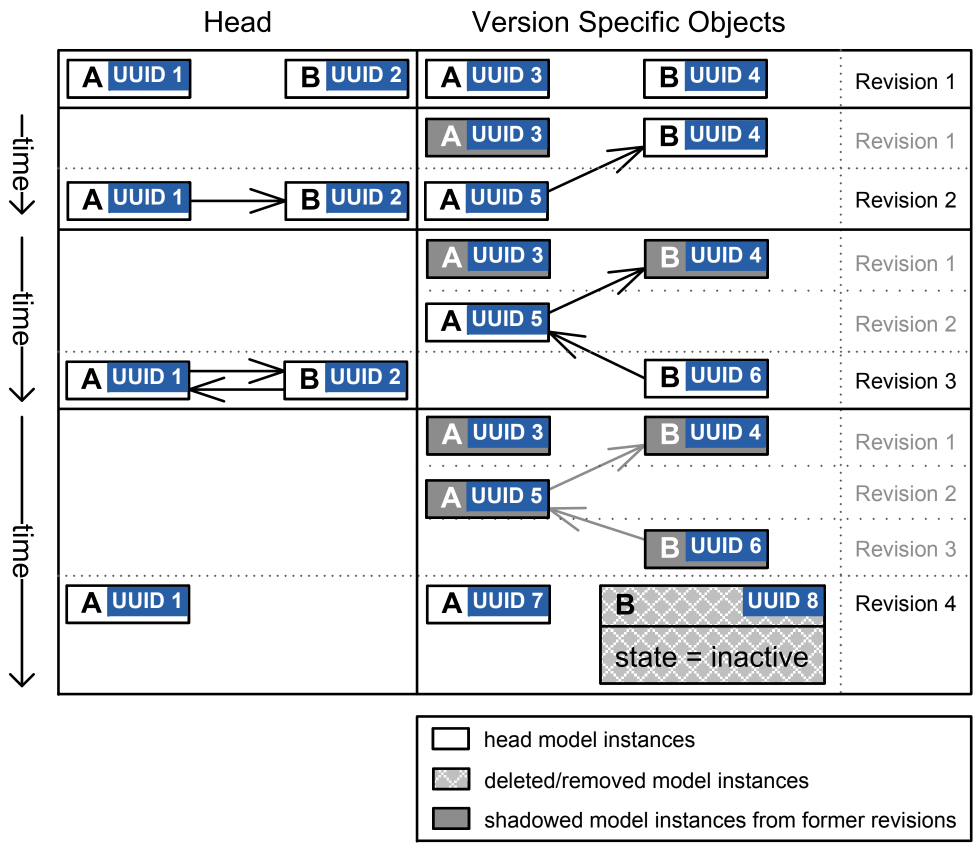


Figure 3.6: Transparent Model Versioning for UUID-Based Model Repositories

updated. Note that a new version-specific object is created that references `B`, *shadowing* the model element from the previous revision. That is, for obtaining a certain revision of a model only the most recent model element until the revision is considered and the older revisions are not; thus, we say, they are shadowed. In Revision 3 a new reference from `B` to `A` is added. Similarly to the previous change, `B` is updated. Note that in the version-specific objects of that revision the reference from `A` still points to the now shadowed `B` from Revision 1. This is no problem however, if for obtaining a model of a specific revision the reference is updated to the most recent version of `B`. Finally, the last change removes `B` from the model. This implies that also the reference from `A` is removed. Therefore `A` is updated and `B` is removed. The latter is realized by introducing a new `B` that shadows former instances and explicitly is marked as deleted (`state = inactive`).

For retrieving a specific model version an algorithm is applied on the version-specific objects. This is shown in Algorithm 4 that describes how a specific version of a model can be calculated from the version-specific objects. First the various model elements of a model in a specific version are retrieved (Line 5). Only model elements that have not been removed (Line 8) are registered (Line 9). The returned model equals to the formerly version-independent objects for the revision. For this, the version-specific elements are transformed to version-independent elements (Line 7) and the UUIDs of the references are updated (Line 15).

Algorithm 4: Reconstructing a Revision of a Model from Version-Specific Objects

```
   Input: UUIDs, revision $\in$ Revisions
   Output: Model
1  begin
2      model $\longleftarrow \emptyset$;
3      vsid2viid $\longleftarrow \emptyset$;
4      for uuid $\in$ UUIDs do
5          element $\longleftarrow$ retrieve(uuid, revision);
6          vsid2viid.put(getVSID(element), uuid);
7          makeVI(element);
8          if $\neg$ isInactive(element) then
9              model.put(uuid, element);
10     for reference $\in$ getReferences(model) do
11         vsid $\longleftarrow$ getUUID(reference);
12         viid $\longleftarrow$ vsid2viid.get(vsid);
13         if $\emptyset$ = viid then
14             viid $\longleftarrow$ retrieveVIID(vsid);
15         setUUID(reference, viid);
16     $\longleftarrow$ model;
17 end
```

3.5 MORSE Services

For supporting both model evolution and the use of models at runtime we propose a model-driven and service-based approach for services to dynamically work with models in a SOA.

For this, we automatically generate MORSE services for managing models as depicted in Figure 3.7. Usually, a domain expert starts to design a domain model (Step 1), that is, a meta-model with concepts of a certain domain. Next, a technical expert refines the model (Step 2), e.g., by enriching the model with technical details as needed in further model-driven process steps. Domain-specific languages (DSLs) can assist the different stakeholders to formulate the model. Finally, in order to support the use of the resulting model in a SOA, MORSE services are

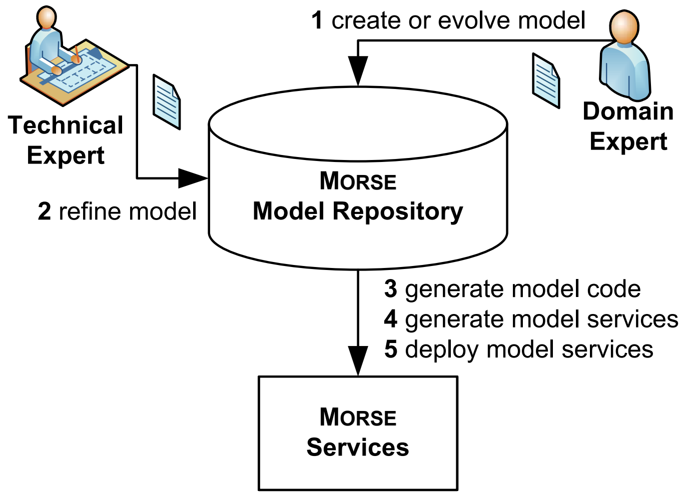


Figure 3.7: Generating MORSE Services for Models

automatically generated and deployed (Steps 3-5) that can be used by other services for sharing, storing, and retrieving models [8].

In order to support the development of a monitor MORSE can also automatically generate ready to use service requester agents that interact with the MORSE services. This is demonstrated in Figure 3.8 (Step 2). After deployment (Step 1), the exposed MORSE services can be used by the monitoring service, which is a model-aware service, for storing and retrieving models (Step 4). Note that, while model-aware services can be manually developed by a service developer as

[8]For model transformation we use Frag, a dynamic programming language that specifically is designed to support MDE. For model transformation we particularly profit from Frag's dynamic class concept and structural reflection using introspection options (cf. [185]) that allow for efficient exploitation of models. The transformation has been realized as follows: First we transform the Ecore model to a Frag model. For model-to-code transformation we have then implemented code generators that produce corresponding interfaces and concrete implementations.

For demonstrating and safeguarding the functionality of MORSE repositories we have developed executable tests. Like the implementation of MORSE these tests are model-driven and automatically generated. In a test class, a set of objects with their properties and references – as derived from the model – is first populated and filled with random values according to their type. Various tests follow that check the functionality of the implementation, such as the creation of MDE projects, artifacts, and their associations. For maximizing the coverage of these tests, we made use of a coverage-based testing tool [181, 56] during development. Besides these model-driven tests, additional tests can be supplied. Also the template itself can be extended.

Finally, we have coupled MORSE repositories with a system for CI [155, 100]. A CI server creates and tests a build by executing the appropriate MDE workflow of a project. As Maven [157] is commonly used by CI systems for building and testing we generate appropriate project object model (POM) files for MORSE repositories and the registry. Besides a component test of MORSE repositories and the registry, the CI server runs automatic system tests with other MDE components that it manages: when changes to the MORSE repository, the registry, or other MDE components occur, the CI server performs all necessary tests and in case of a failure gives rapid feedback to the developers. This way, evolution of the MORSE repository or other MDE components is eased, as errors due to the integration of components are detected early and can be fixed timely.

depicted in the figure (Step 3), they may be also automatically generated from models.

If a meta-model evolves, MORSE generates appropriate services and service requester agents. This simplifies maintenance of the model-aware services that can also automatically be instructed to work with the new version.

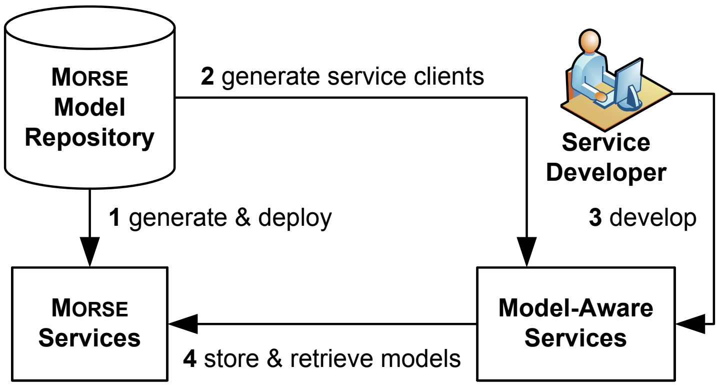


Figure 3.8: Supporting Model-Aware Services with Service Clients

We will now describe the MORSE services that provide retrieve and storage functionalities for models in a SOA and realize the transparent UUID-based model versioning as explained in Section 3.4.2. These services are automatically generated from meta-models and exposed as XML [9] and representational state transfer (REST)ful [10] Web services. Model-aware services can thus choose between these implementations. Note that in addition to a service implementation MORSE also generates service requester agents, i.e., proxies, (see also Step 2 of Figure 3.8) for using the MORSE services.

All of the generated software modules are organized as Maven [157] projects and distributed using a Maven repository [116]. In this way, service developers such as displayed in Figure 3.8 are provided with the client software modules for integrating with the MORSE services and are thus supported for using models at runtime. With Maven it is very easy to setup a dependency (see also Listing 6.3) that automates the retrieval and use of required modules.

With the model-driven generation, the deployment of services, and the distribution of service requester agents the major part for supporting a new or evolved model at runtime is realized and fully automated. At present, for this, MORSE supports two different modeling technologies with Ecore – which is the EMF [162] implementation of the EMOF M3 meta-meta-model (cf. [131]) – to be natively supported. Models that conform to other M3 models can be supported with a model-to-model transformation and a mapping to Ecore if necessary as well. Please note that

[9] realized with the Java API for XML - Web Services [104] (JAX-WS)
[10] realized with the Java API for RESTful Web Services [75] (JAX-RS)

while for a technical realization MORSE itself builds on EMF, MORSE tries not to place any restrictions on the use of different model technologies. MORSE services and its operations in particular are agnostic to modeling technologies and serialization formats. Support for further modeling technologies can be realized in the MORSE services if necessary. As a result, model-aware services are not limited to work with EMF but can use different model technologies as well.

Table 3.2: MORSE Service Operations

Response	**Operation**	**Description**
boolean	exists	does a model with a UUID exist?
boolean	isHead	is the object (specified by UUID) version-independent?
UUID[]	list	returns the VIIDs of all models
UUID[]	versions	returns all VSIDs of a model
<*C*lass>[]	query	search for models; support of various query parameters
<*C*lass>	retrieve	a model is retrieved by UUID
UUID	create	a VIID is returned
UUID	update	a VSID is returned
UUID	delete	a VSID is returned
UUID[]	list<*R*ole>	returns the UUIDs for a role
UUID	add<*R*ole>	a VSID is returned
UUID	remove<*R*ole>	a VSID is returned
UUID	clear<*R*ole>	a VSID is returned

For each class of a meta-model a service is generated with basic operations such as `create`, `retrieve`, `update`, and `delete` (CRUD) for the management of models as listed in Table 3.2. The UUIDs of all version-independent models can be obtained by calling the `list` operation. Similar operations are generated for the management of references.

The `exists` operation checks if a model for a provided UUID is found in the repository. Whether a UUID is a version-independent or version-specific UUID can easily be checked using the `isHead` operation. The `versions` operation returns the various version-specific UUIDs of a model. The `query` operation returns a set of serialized [11] models that match specified query parameters. These parameters support the various Java Persistence Query Language [54] (JPQL) clauses such as `JOIN`, `WHERE`, or `ORDER`. For *pagination* (cf. [54]) also an index for the first result can be specified as well as the number of maximum results returned.

[11]The EMF supports XMI [135] and Extensible Markup Language [29] (XML) [29] for the serialization of Ecore-based models. Support for other modeling technologies and serialization formats can be added as mentioned.

3.6 Related Work

The MORSE approach particularly focuses on the management of models of service-based systems and their accessibility during runtime. For this reason, a model repository with versioning capabilities is deployed (see Section 3.4.2). It abstracts from modeling technologies and its UUID-based implementation allows for a straightforward identification of models and model elements. There are a number of related model repositories. In Table 3.3 we list their methods of identification for models and model elements. Table 3.4 indicates supported modeling technologies and the smallest UV. Finally Table 3.5 compares navigation and search capabilities of the different model repositories. In the following we will compare to the related work and to the data from the tables.

Table 3.3: Identification of Model and Model Elements in Model Repositories

Repository	**Model Identification**	**Model Element Identification**
AMOR [9, 30]	URL	ID
AtlanticZoo [12]	URL	NO
CDO [160]	URL	URI-Fragment
EMFStore [112, 111, 110]	ID	ID
MDR [122]	ID	URI-Fragment
ModelBus [151, 7, 79]	URL	NO
MORSE	UUID	UUID
Odyssey-SCM [128, 137]	ID	URI-Fragment
Odyssey-VCS 2 [127]	ID	URI-Fragment

The Adaptable Model Versioning [9] (AMOR) model repository, the successor of the ModelCVS [106] and SMoVer [8] model repositories, has a focus on the versioning aspect of model management (see also [10]), e.g., for the conflict resolution in collaborative development (cf. [31]). For this reason the smallest UV can be set to model elements. Models in AMOR are identifiable by Uniform Resource Locator [77, 22]s (URLs). In addition IDs are assigned to model elements. While these are unique across models they are not over time, i.e., model elements from different versions of a model contain the same ID. AMOR builds on top of EMF, focuses on the design time and addresses important research questions in the field of conflict detection and resolution.

The AtlanticZoo [12] is a simple, web-based model repository, thus models are accessible via URLs. It aims at constituting a recognized repository for open-source models. For this purpose and for maximizing potential usage of contributed models they are automatically transformed from and into diverse languages such as Ecore, Kernel Meta Meta Model [99, 165] (KM3), Web Ontology Language [17] (OWL), or UML. Versioning is not in focus of this repository that rather can be characterized as a collection of models. These are stored in a serialized form. For this reason the repository is agnostic to modeling technologies and is ignorant of model elements. Thus, no model element identifiers exist that are supported by the repository.

Table 3.4: Modeling Technologies and Units of Versioning in Model Repositories

Repository	**Modeling Technology**	**Unit of Versioning**
AMOR	EMF	ANY
AtlanticZoo	ANY	model
CDO	EMF	$M2$ class instance
EMFStore	EMF	ANY
MDR	MOF 1.4 [132]	ANY
ModelBus	ANY	model
MORSE	ANY	$M2$ class instance
Odyssey-SCM	MOF 1.4	ANY
Odyssey-VCS 2	EMF	ANY

The Connected Data Objects model repository [160] (CDO) is a server-client framework for EMF models. In EMF a model element is identifiable within a model [12] via a so called Uniform Resource Identifier [77] (URI)-fragment. Usually, although pluggable in CDO, a RDBMS is used as a persistence backend (e.g., with Teneo [167]) in which case the smallest UV is at an $M2$ class instance level. The CDO framework establishes a CDO protocol on top of the Net4j [166] communication framework and also aims to support the execution of server-side queries.

EMFStore [112, 111, 110] is a model repository for the Eclipse integrated development environment (IDE) that employs operation-based change tracking, conflict detection, and merging. As a result it is specific to EMF models that Ecore class instances need to inherit an EMFStore class in order to be tracked and managed by EMFStore. While EMFStore does not support complex queries, (server-side) model navigation may be realized. IDs are used for identifying models and model elements. These are unique across the space of models and model elements but not across time, i.e., models from previous versions contain the same IDs. The UV in EMFStore due to its operation-based approach can be of any size.

The NetBeans metadata repository [122] (MDR) was a MOF 1.4 compliant model repository for the NetBeans IDE that is not actively developed and maintained any more. It was used as a persistence backend by Odyysey-SCM (see below).

[12]A *Resource* (identifiable by a URL), to be more precise in terms of EMF

ModelBus [151, 7, 79] is a model-based tool integration framework. It addresses the heterogeneity and distribution of model tools and realizes transparent model update. Designed as an open environment, ModelBus focuses on integrating functionality such as model verification, transformation, or testing into a service bus. It is agnostic to modeling languages and uses a VCS as persistence backend. Thus, models are stored in their serialized forms. As a consequence the UV is the entire model and no navigation or search capabilities exist. Models in ModelBus are identified by URLs but no identifiers exist for model elements.

Table 3.5: Model Navigation and Search in Model Repositories

Repository	**Model Navigation**	**Complex Search**
AMOR	NO	NO
AtlanticZoo	NO	NO
CDO	YES	YES
EMFStore	YES	NO
MDR	NO	NO
ModelBus	NO	NO
MORSE	YES	YES
Odyssey-SCM	NO	NO
Odyssey-VCS 2	NO	NO

Odyssey-SCM [128, 137] and Odyssey-VCS 2 [127] identify models using XMI IDs. Model elements are identified with additional URI-fragments. While Odyssey-SCM used MOF 1.4 [132], Odyssey-VCS 2 builds on EMF. Great focus is dedicated to the versioning aspect and conflict detection. For this the authors defined the terms unit of versioning (cf. [137, p.4]) (UV) and unit of comparision (cf. [137, p.3]) (UC) and make these customizable for the SCM of UML model elements. While complex model search scenarios and navigation are not supported by the repository, model navigation is at least possible for source and destination models of model transformations in Odyssey-MEC [51] through exogenous "records of transformation".

In contrast to the mentioned model repositories and model-based tool integration frameworks, Moogle [118], a model search engine, realizes an inverse approach of indexing and potentially managing models. It allows for complex queries and can help finding relevant models for MDD projects during design time. As such it is not (yet) suited for our purposes that target runtime systems.

All mentioned model repository approaches do not provide transparent UUID-based model (element) versioning capabilities, a central contribution of our work. From the compared repositories, MORSE is the only model repository that allows models and model elements to be identified by means of simple UUIDs, i.e., without the need of multiple identifiers. We consider the unique identification of models and model elements as important in regard to the runtime use of models. This is because different runtime systems may require different versions of a model or model element. Hence, model evolution is harder in these approaches.

In contrast to most model repositories, the MORSE repository abstracts from technologies and focuses on MDD projects. This is, the MORSE repository comes with explicit support for the management of MDD projects and supports the MDD process (e.g., through a navigation compatibility check as presented in the previous chapter or a generation service), something that is not supported by many other repositories (e.g., workflows, that cover processes of MDD, have to be defined on top of ModelBus).

MORSE is the first model repository that specifically targets at integration with runtime services. Other work mainly focuses on the design time, e.g., in order to support the MDD process. With our MORSE approach however we aim at adopting models at runtime, i.e., using models beyond the model-driven generation step. Thus, MORSE focuses on runtime services and processes and their integration, e.g., through monitoring, with the repository and builds on the simple identification for making models accessible at runtime.

None of the mentioned model repositories offers an integration scheme for runtime events as we will show in Chapter 6 or the automated model generation and deployment capabilities. As shown in this chapter, however, for supporting models at runtime, if model evolution is possible some similar capabilities would be needed. However, our approach is general enough and not limited to MORSE: It should be possible to extend all the model repository approaches that we mentioned using a frontend that extends them with transparent UUID-based model versioning capabilities and/or at least UUID-based identification and provides a model-driven component for runtime generation and deployment.

3.7 Summary

We have presented the Model-Aware Service Environment [82, 87, 86] (MORSE) for facilitating services to dynamically reflect on models. For this purpose we have proposed a model repository that manages MDD projects and supports the identification of and reflection on models, model elements, and model relationships.

Many features of CDEs, such as instant messaging, discussion forums, or voting can be applied to any domain. In contrast to these, *information management* is more domain dependent: By contextualizing information, a CDE gains an understanding on its type, semantics, dependencies, and properties. A repository that manages these information, allows for tailored information retrieval (IR). MORSE repositories with their conceptual model of MDE projects and artifacts allow for this kind of IR, and, thus, address the information management challenge of CDEs with regard to MDE.

The MORSE repository stores and manages models, model elements, model instances, and other MDD artifacts. It offers read and write access to all artifacts at runtime and design time. Moreover, it stores relationships among the MDD artifacts, e.g., model relationships such as instance, inheritance, and annotation relations. Information retrieval (i.e., the querying of models) is supported for all MDD artifacts and relationships via service-based interfaces, which ease the integration of MORSE into service-oriented environments. For reflectively exploiting the relationships of MDD artifacts, the MORSE IR interface provides various methods, too. An example of such a reflective relationship access is to retrieve all model instances for a model. By traversing the relationships and exploiting properties, queries can be constructed in an interactive, stepwise manner. For performance reasons queries can also be combined and executed as complex queries (e.g., *return all projects that models annotate model instances of a certain model*). We addressed the need for the management of different model versions with a transparent versioning in MORSE. This way, models can be manipulated at runtime of the client system with minimal problems regarding maintenance and consistency. New versions and old versions of the models can be maintained in parallel, so that old model versions can be used until, e.g., all their model instances are either deleted or migrated to the new model version.

For facilitating services to work with models at runtime, MORSE services are automatically generated and deployed for domain concepts that are expressed as EMF models. That is, for every concept a service is generated with basic *create*, *retrieve*, *update*, *delete*, and *query* operations. For the different relations between concepts appropriate *add* and *remove* operations are generated in addition for associating and deassociating role instances.

These services, that realize the transparent versioning, can be used by other services, that we called model-aware services. For easing the development of such services MORSE can also automatically generate appropriate service requester agents.

The presented services of the model repository are not only of interest for the runtime but also for design time components. With appropriate tool support, the MORSE repository constitutes a common space for developers in a distributed environment for storing and managing MDD projects and models that can be consulted by the runtime as well. This way, MORSE bridges the gap between the MDE phases and unifies the use of models in MDE. In the following chapter we discuss model-aware systems that may interact with MORSE for dynamic model look-up and reflection.

CHAPTER 4

Model-Aware Systems

Mater artium necessitas. [1]
— WILLIAM HORMAN [2]

Contents

[1]*Necessity is the mother of invention.*
[2]An original author is unknown.

In a number of scenarios, services generated using a model-driven development (MDD) approach could benefit from "reflective" access to the information in the models from which they have been generated. Examples are monitoring, auditing, reporting, and business intelligence scenarios. Some of the information contained in the models of a service can statically be generated into its source code. In a distributed and changing environment this approach is limited, however, due to the fact that models and their relations evolve after the generation and deployment of a service. For example, the current model of a service might be different from the deployed version of the service. Thus, for retrieving an up-to-date version of a model, dynamic model look-up during runtime becomes necessary.

We call systems, services, and processes that contain, emit, or use model traceability information for model look-up and reflection *model-aware*. These systems may interact with the previously presented Model-Aware Service Environment [82, 87, 86] (MORSE) at runtime and can profit from its reflective functionalities. In this chapter we illustrate how model-aware services can interact with the repository and what information they may expose. Also we will show how they can be used by other services and how they can be created. Finally, we demonstrate an integration of process-driven service-oriented architectures (SOAs) with model-aware services, the foundation for model-aware monitoring of business processes that we will present in the following chapter.

4.1 Model-Aware Services

In a model-driven SOA, the services are in large parts generated from models. To facilitate monitoring, governance, and self-adaptation of SOAs, the information in these models can be used by services that monitor, manage, or adapt the SOA at runtime. If a service for monitoring, management, or adaptation in an SOA is dependent on models, and the meta-model changes, usually the service needs to be manually adapted to work with the new version, recompiled, and redeployed. This manual effort impedes the use of models at runtime. Besides the navigation compatibility check (see Section 2.4), we have presented the automatic generation of MORSE services (see Section 3.5) to address this problem. Model-aware services, that we introduce in this section, are supported with these services from the Model-Aware Service Environment that manages models and offers transparent versioning of them (see previous chapter). MORSE uses the model-driven approach to automatically generate and deploy MORSE services that can be used by the model-aware services to access models in the correct version. In this way, monitoring and adaptation in SOAs can be better supported and the manual effort to evolve services for monitoring, management, or adaptation, which are based on models at runtime, can be minimized. To the best of our knowledge, our approach is the first approach that makes this link between services and the models from which they are generated at runtime.

4.1.1 Interaction with MORSE

Model-driven, model-aware services can retrieve the MORSE objects from which they have been generated. This is achieved by embedding the Universally Unique Identifier [113, 98]s (UUIDs) of the objects into the services, such as the UUID of the build as well as UUIDs of corresponding models, model elements, or transformations. At runtime, the services can access these UUIDs and retrieve the MORSE objects from the repository. A model-aware service typically reflects on the information and applies some logic for its further execution or uses the information for performing model-transformations.

Figure 4.1 illustrates a sequence diagram of a model-aware service. After a project and MDD artifacts have been created and checked into the repository, a build for the project is initiated by a service client. The builder service retrieves the artifacts and generates a model-aware service, weaving corresponding UUIDs into the code for traceability. Afterward, it is deployed to a Web service framework. Next, a requester agent invokes the service and causes it to interact with the repository. For the models from which it has been generated, it needs, e.g., to discover and consider new model relations such as new annotations. Therefore it passes the embedded UUIDs of its models to the information retrieval (IR) interface of the model repository and requests for the model relations. These are then retrieved and evaluated. Relevant model relations are identified and the related models are requested. Finally, the models are processed and a response is issued to the service requester.

Please note that the MORSE builder in Figure 4.1 uses the MORSE repository to obtain the models of the model-aware services in order to generate code for them. This sequence diagram hence illustrates how an MDD tool (in this case the generator of MORSE) can make use of the MORSE architecture in the same way as other components querying models, such as monitoring components.

4.1.2 Informational Operation

We have seen how model-aware services can interact with the repository, e.g., after invocation. Besides this, model-aware services may also offer information on them to other services (see Λ in Figure 3.1), i.e., disclose the UUID of their `MBuild`. The caller can use the returned UUID from the model-aware service for requesting further information from the MORSE repository such as the `MProject` or the `MArtifacts` that have been used for the `MBuild`, i.e., for generating the model-aware service.

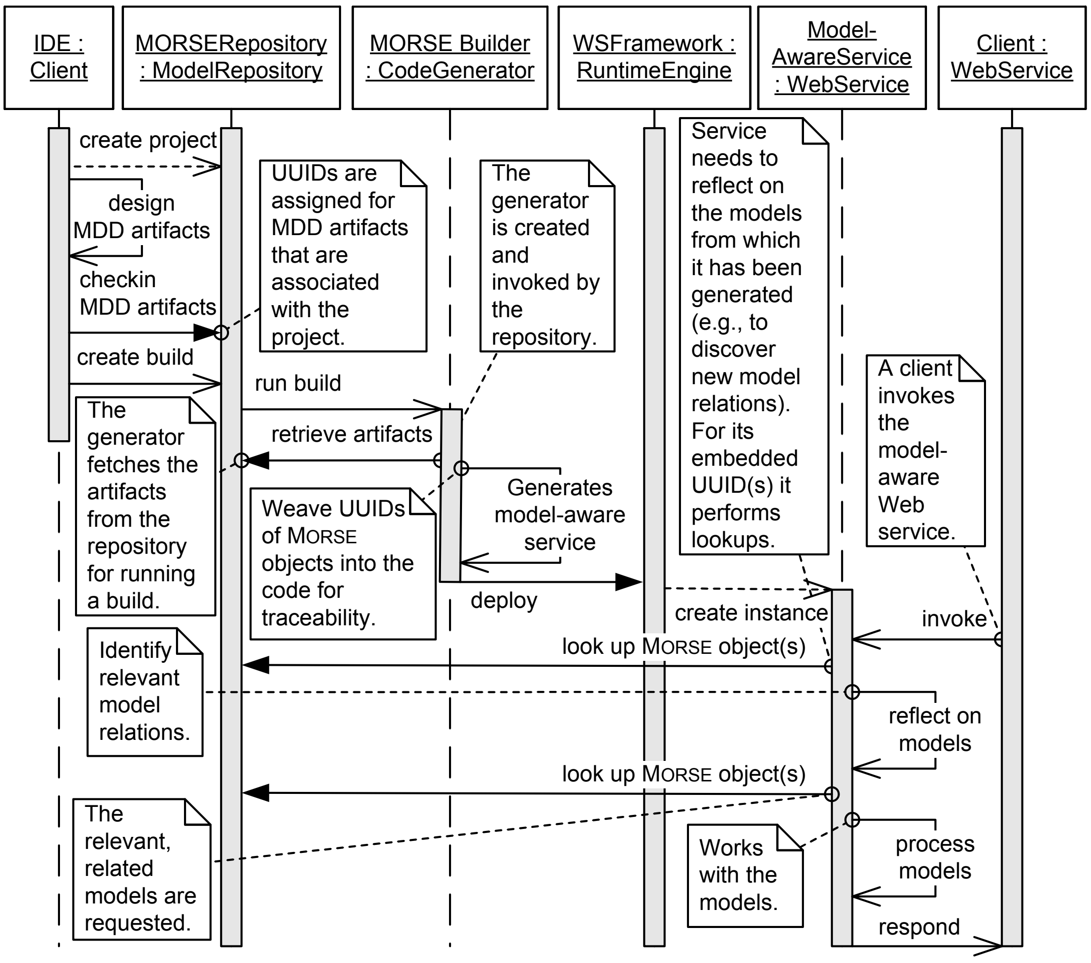

IDE : Client
MORSERepository : ModelRepository
MORSE Builder : CodeGenerator
WSFramework : RuntimeEngine
Model-AwareService : WebService
Client : WebService
create project
design MDD artifacts
checkin MDD artifacts
create build
run build
retrieve artifacts
Generates model-aware service
deploy
create instance
invoke
look up MORSE object(s)
reflect on models
look up MORSE object(s)
process models
respond
UUIDs are assigned for MDD artifacts that are associated with the project.
The generator is created and invoked by the repository.
Service needs to reflect on the models from which it has been generated (e.g., to discover new model relations). For its embedded UUID(s) it performs lookups.
A client invokes the model-aware Web service.
The generator fetches the artifacts from the repository for running a build.
Weave UUIDs of MORSE objects into the code for traceability.
Identify relevant model relations.
The relevant, related models are requested.
Works with the models.

Figure 4.1: Sequence Diagram of a Model-Aware Service

4.1.3 Creating Model-Aware Services

The MORSE builder supports the generation of the presented types of model-aware services, i.e., it creates Web Services Description Language [40] (WSDL) or Web Application Description Language [74] (WADL) interfaces and Java implementations, as follows:

- For any build of a MDD project, a dedicated, standalone Web service can be generated that provides information on the build. For the endpoint of such a Web service that shall contain a `uuid` operation that returns the UUID of the `MBuild` of the service we propose a Uniform Resource Identifier [77] (URI) that ends with `/mas/MBuild` as a naming convention.

- For generating model-aware services, templates can be reused in projects for extending the interface with the desired operation, for embedding the UUIDs to the service, and for generating the service requester implementation for interaction with the repository.

4.2 Integrating with Model-Aware Services

A service or process may integrate with and use model-aware services (see also B in Figure 3.1). A model-aware service can support model-driven systems in the sense that it can look-up and work with the MDD artifacts from which they have been generated. During runtime, it receives events from these systems that contain MORSE identifiers and queries the model repository.

4.2.1 Enriching Process-Driven SOAs with Traceability

As an example let us consider processes with the Business Process Execution Language [139] (BPEL) as a target technology for a process-driven SOA. Such processes can be generated from platform-independent, conceptual models as we will illustrate in the following chapter. The model repository manages the respective MDD projects and artifacts. In general, processes and process engines do not interact with the model repository or model-aware services. However, they can *integrate* with model-aware services in the sense that the latter receive events from the engine that hold the UUIDs of MORSE objects. Thus, these UUIDs have to be supplied as traceability information to the process and during process execution UUIDs have to be emitted that correspond to the process or process activities.

For example, BPEL extensions provide a standard way to realize this (cf. Listing 6.1 in Section 6.3). The `traceability` element – defined in the Extensible Markup Language [29] (XML) Schema [170, 25] shown in Listing 4.1 – that indicates the UUID of the `build` as an attribute, is a sequence of `rows` that maps BPEL elements to the `uuids` of corresponding MORSE objects. The XML Path Query Language [21] (XPath) is chosen as the default query language for selecting the XML elements of the BPEL code. For extensibility, an optional `queryLanguage` attribute, that has the same semantics as in BPEL (cf. [139, Section 8.2]), can specify an alternative query language or XPath version.

```
<?xml version="1.0" encoding="UTF−8" standalone="no"?>
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:morse="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability"
    xmlns:xmi="http://www.omg.org/XMI"
    targetNamespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability">
  <import namespace="http://www.omg.org/XMI"
```

```
      schemaLocation="http://www.omg.org/spec/XMI/20071213/XMI.xsd"/>
  <complexType name="Traceability">
    <choice maxOccurs="unbounded" minOccurs="0">
      <element name="row" type="morse:TraceabilityRow"/>
      <element ref="xmi:Extension"/>
    </choice>
    <attribute ref="xmi:id"/>
    <attributeGroup ref="xmi:ObjectAttribs"/>
    <attribute name="build" type="string"/>
  </complexType>
  <element name="traceability" type="morse:Traceability"/>
  <complexType name="TraceabilityRow">
    <choice maxOccurs="unbounded" minOccurs="0">
      <element name="uuid" nillable="true" type="string"/>
      <element ref="xmi:Extension"/>
    </choice>
    <attribute ref="xmi:id"/>
    <attributeGroup ref="xmi:ObjectAttribs"/>
    <attribute name="queryLanguage" type="string"/>
    <attribute name="query" type="string" use="required"/>
  </complexType>
  <element name="traceabilityRow" type="morse:TraceabilityRow"/>
</schema>
```

Listing 4.1: MORSE Traceability XML Schema

Note that this `traceability` matrix can annotate any XML-based target code and can often be supplied as an inline extension [3] and does not have to be defined in a separate file. As a consequence, our approach is not limited to BPEL but can directly be applied to other XML and Web service based technologies and standards. The traceability information can also be applied to programming languages such as Java or C#, e.g., as annotations to classes, interfaces, methods, and parameters. These annotations can be added to the source code (cf. [33]) or can be realized exogenously in an annotation file that decorates annotated classes.

In our BPEL example, during generation time, the MORSE builder (cf. Figure 3.2 and Section 4.2.2) weaves UUIDs of the MORSE objects into BPEL code (see A in Figure 3.1). At deployment time, the BPEL engine needs to support the BPEL extension, i.e., for the namespace that is used for the MORSE traceability extension, there is an implementation at the BPEL engine in place. At runtime, this extension submits events that contain the identifiers of, e.g.,

[3]Supposed that such extensibility is provided with an `any` element in the XML schema.

the process or process activities, an event type, and optional further properties. Some events of interest are process instantiation and termination and pre-events and post-events for the execution of activities. Finally, the events are received by model-aware services that look-up the MORSE objects for, e.g., monitoring, auditing, reporting, or business intelligence scenarios. We will cover the topic of monitoring in Chapter 6.

4.2.2 Creating Services and Processes that support Model-Aware Services

For services and processes that rely on model-aware services, UUIDs need to be supplied for services or processes as well. During generation time, the MORSE builder creates a traceability mapping that can be embedded into XML documents as demonstrated. Although we have shown an integration with model-aware services using model-driven BPEL processes, this approach can similarly be applied to different technologies and frameworks, e.g., message interceptors for Web services.

4.3 Deployment Service

The MORSE builder service is complemented with a deployment service that supports the deployment of business processes and Web services. For the deployment of business processes we have developed a validation/deployment/execution (VDE) framework (cf. [109, 84]). This framework permits generic validation, (un-)deployment, and execution of BPEL processes. In this way flexibility concerning the BPEL engine of choice is given and a unique interface for validation, deployment, and execution can be used.

In order to establish a software architecture that fulfills the requirement to include additional BPEL engines without having to recompile the framework, a plug-in [4] architecture was chosen using the abstract factory software pattern [68, 67]. In the remainder of this section we describe the basic functionality of the VDE framework that interface is described in Table 4.1.

Validation Similarly to the deployment (see below), also a validation of business processes, can be realized in a plug-in fashion. A foundational unifying framework based on the π–calculus that could be used for validating various properties of a business process within the VDE framework has been developed in [117] and [124]. Alternative semantic validation may also be realized, e.g., after transforming BPEL to petri nets [182].

[4] Providers for ActiveBPEL [2] and the Apache Orchestration Director Engine [158] (Apache ODE) have been implemented.

Table 4.1: Service Operations of the VDE Framework

Method	**Description**
create archive	create an engine specific process archive
deploy	deploy an archive / BPEL process
execute	execute a BPEL process operation
get deployment information	general information on the provider
list deployments	list all deployed processes
load deployments	load an existing deployment
validate	validate the BPEL process
undeploy	undeploy the archive / BPEL process

Archive Creation An important functionality for simplifying and unifying the deployment of processes realized by the VDE framework is the creation of a BPEL archive. Different BPEL engines require such an archive for the deployment of business processes. However there are differences in the required structure: In order to generate an appropriate archive for the BPEL engine of choice typically specific descriptors have to be provided together with the BPEL/WSDL files within a compressed file.

Deployment After a BPEL archive has been generated, deployment can take place. Deployment of a BPEL archive can be realized in various ways, e.g., by copying the archive to a specific destination on the file system or by invoking a Web service.

Execution After deployment, instances of the process may be initialized and executed (e.g., by invoking a `receive` activity). For this, the VDE framework particularly supports the In-Only message exchange pattern [39] (MEP) for the instantiation and execution of business processes.

4.4 Case Study

In order to demonstrate the applicability of the presented approach, we explain a case study. In this case study a European telecommunication company offers rich multimedia services to customers. Particularly, the company's customer can subscribe to content such as video or audio streams and files, i.e., the customer can, e.g., download tracks from music albums and watch movies.

In this context, licensing information on the content constitutes a crucial issue: (under which conditions) is the customer allowed to access a resource? Often, e.g., broadcasting content can only be obtained if a user executes the request to access such content within the country of the broadcasting company. Other possible restrictions are the contract and status of the customer. Similarly, payment depends on various factors such as the price of the requested content, the conditions of the customers contract and/or of effective special offers.

The telecommunication company decided to apply MDD technologies for its services. Therefore, it designed various models, e.g., for licensing and payment information. When the company modifies its business models for multimedia content, e.g., introduces a special offer, or if it changes the licensing information, it creates or modifies models accordingly. For enabling runtime services to dynamically work with these models, the company employs MORSE, i.e., models are stored within the model repository and the SOA contains model-aware services that interact with the repository. As a result, the company is able to address the following issues:

- The current price for the content as well as effective conditions such as originating from valid special offers or the customers contract conditions are considered for calculating the price. A special offer can simply be introduced as a new model relation, i.e., the service does not have to be modified or redeployed.
- Access is granted as specified in the effective licensing model.
- For analyzing customer services the corresponding processes are monitored and related to their originating models.

Using the builder (see Sections 4.1.3 and 4.2.2) and deployment (see Section 4.3) services the `ConsumeMultimediaProcess` is available at the process engine. An instance is created when the user wants to download some multimedia content. This process invokes an `AccountingService` and a `LicensingController` service and returns detailed information on payment and licensing to the user for acceptance. Both services retrieve the model instances from the repository and apply an algorithm. If necessary, also the algorithms can be stored as models and can dynamically be retrieved and executed by the services. We will demonstrate such a model-aware service in more detail by focusing on the payment service. Its service invocation from the process is shown in Figure 4.2.

The orchestrating process also notifies a monitor by transmitting the UUID of the model from which it has been generated. This is shown in Figure 4.3. From the process models BPEL code is generated with a BPEL extension that notifies the monitor at invocation, as explained in Section 4.2. A monitor collects information from various processes and process versions and correlates them for generating statistical reports (containing, e.g., process versions, number & time of invocations, and duration).

After the `AccountingService` is invoked, it calculates a price with the effective payment and content models according to the conditions of the contract and of effective special offers.

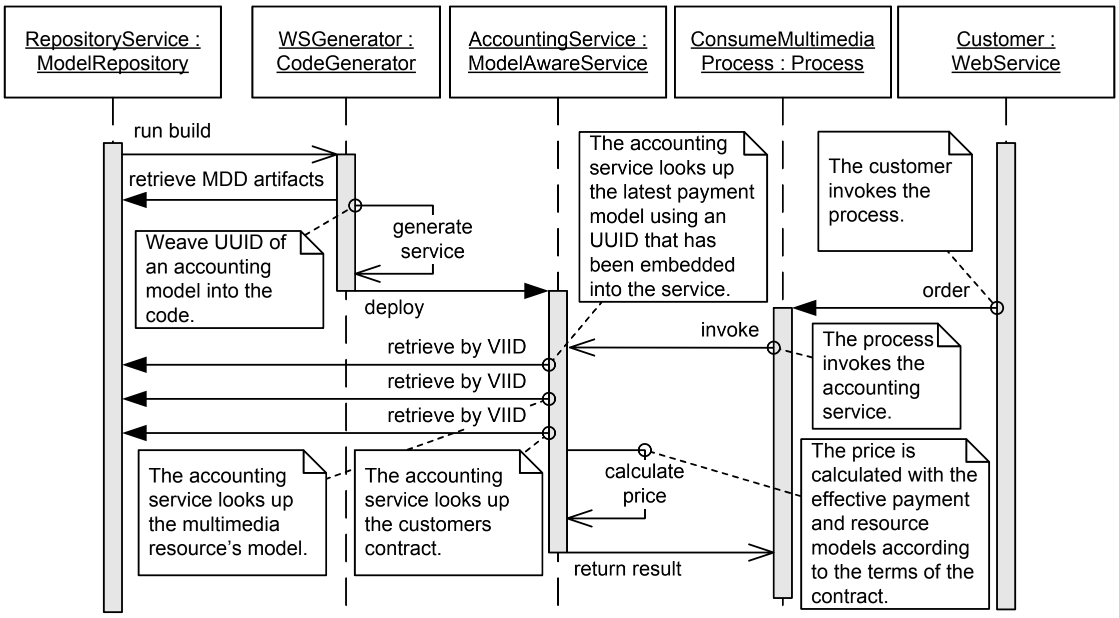

RepositoryService : ModelRepository
WSGenerator : CodeGenerator
AccountingService : ModelAwareService
ConsumeMultimedia Process : Process
Customer : WebService
run build
retrieve MDD artifacts
generate service
Weave UUID of an accounting model into the code.
deploy
The accounting service looks up the latest payment model using an UUID that has been embedded into the service.
The customer invokes the process.
order
invoke
The process invokes the accounting service.
retrieve by VIID
retrieve by VIID
retrieve by VIID
The accounting service looks up the multimedia resource's model.
The accounting service looks up the customers contract.
calculate price
The price is calculated with the effective payment and resource models according to the terms of the contract.
return result

Figure 4.2: Accounting Service – Sequence Diagram

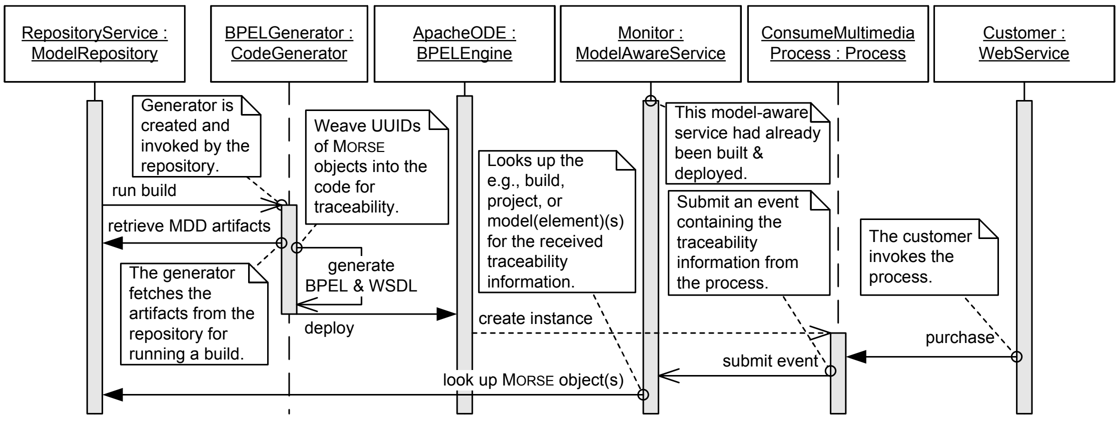

RepositoryService : ModelRepository
BPELGenerator : CodeGenerator
ApacheODE : BPELEngine
Monitor : ModelAwareService
ConsumeMultimedia Process : Process
Customer : WebService
Generator is created and invoked by the repository.
Weave UUIDs of MORSE objects into the code for traceability.
Looks up the e.g., build, project, or model(element)(s) for the received traceability information.
This model-aware service had already been built & deployed.
Submit an event containing the traceability information from the process.
The customer invokes the process.
run build
retrieve MDD artifacts
generate BPEL & WSDL
The generator fetches the artifacts from the repository for running a build.
deploy
create instance
purchase
submit event
look up MORSE object(s)

Figure 4.3: Monitoring – Sequence Diagram

The models are retrieved from the MORSE repository (see also Figure 4.1). Figure 4.4 shows models that are processed by Algorithm 5 for calculating the price. Besides customer, content, and purchase information, the models store conditions of a contract and optionally of special offers. For calculating the price, first the effective conditions are determined by applying present special offers. If the flat rate condition is valid, the customer will not be charged for the down-

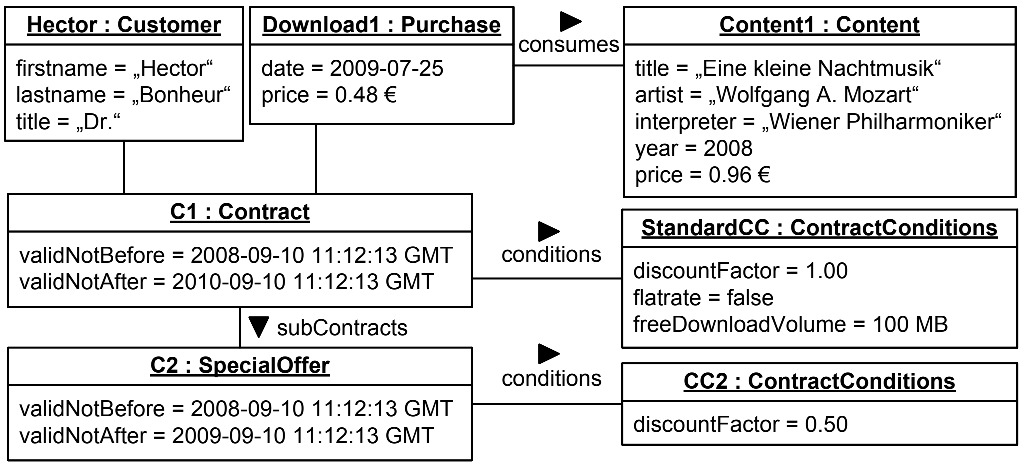


Figure 4.4: Payment Model Instances

Algorithm 5: Payment Algorithm

```
   Input: r ∈ Content, c ∈ Contract
   Output: price ∈ Price
 1 begin
 2     cc ⟵ c.conditions;
 3     for special ∈ c.subContracts→forAll(sc|isValid(sc)) do
 4         applyConditions(special.conditions, cc);
 5     if cc.flatrate then
 6         return 0;
 7     if ∅ ≠ c.purchases→forAll(p|p.date+24h>now() ∧ p.content=r) then
 8         return 0;
 9     if getTotalDownloadVolume(c) < c.conditions.freeDowloadVolume then
10         return 0;
11     else
12         return r.price ∗ cc.discountFactor;
13 end
```

load. Similarly, a customer may download a content that he already retrieved within the last day for free. Otherwise, he will be charged the price of the content by considering a discount if he exceeds his free download volume. After the `ConsumeMultimediaProcess` determined the licensing conditions and calculated the price, the customer is informed and can decide to accept the terms for eventually purchasing the multimedia content. In case access is denied as caused by some licensing condition, the user can be provided with detailed information for understanding the reason. Similarly, he can retrace the payment information by reflecting on the models.

4.5 Related Work

France and Rumpe [65] portray a vision of a model-driven engineering (MDE) research roadmap that classifies and discusses various problems in the field of MDE. The authors distinguish *development* models [5] and *runtime* models, that present aspects of an executing system. They state, however, that – as MDE research matures – this classification may be blurred as both type of models may be used similarly (e.g., in different phases of the MDE process). Although in our work we do not specifically define and utilize runtime models, we do contribute to the universal adoption of models: Model look-up is generically supported (i.e., for all types of models) in model-aware systems through MORSE.

In contrast to the runtime use of models that describe systems (cf. development models [65]), a community of researchers focuses on the use of runtime models that represent views of an executing system: For the generation of middleware configurations, modeling, meta-modeling, and reflection is combined by Bencomo et al. [18]; Two reflective middleware implementations, Open ORB and DynamicTAO, are presented by Kon et al. [103] that allow for the inspection and adaptation of the middleware system; Blair et al. [26] motivate the use of runtime models for dynamic adaptation scenarios (e.g., in ultra-large-scale (ULS) systems); Finally, the idea to use models at runtime is pursued by Bencomo et al. [19] that propose an approach for a runtime reflection in the context of system requirements and employ requirements as runtime entities.

All of these works and our approach focus on runtime reflection on models in runtime systems (e.g., for system adaptation). For this MDE modeling and generation techniques are utilized for creating and adapting systems. One point that distinguishes our approach from the literature is that traceability information is embedded during the generation step into model-driven systems, making them model-aware. As a result model-aware systems can expose their traceability information to other systems and can emit the information within events. Beyond that, and for the reflection on models, they can dynamically retrieve information from the models via *model look-up* in a *distributed* environment, which is an important distinction in our work. In a pull style, model-aware systems can actively query, retrieve, and reflect on models at runtime in a SOA. In contrast, in [19] goal models are synchronized with the architecture in a push style. For scalability reasons we propose to realize such synchronization in a publish/subscribe manner, however. For this the MORSE repository notifies interested parties by exposing a Really Simple Syndication [179] (RSS) feed or by applying an event-driven approach (e.g., in combination with complex event processing (CEP)): In case a model is changed, an event with traceability

[5] I.e., requirements, architectural, implementation, and deployment models

information is raised (submitted to a CEP engine and reflected in the RSS feed) and – using the MORSE services – model-aware services can retrieve the respective models.

Model-aware systems constitute a general approach for dynamic model reflection in a distributed Internet-based environment at runtime. As such it can easily be employed for different scenarios (e.g., in Chapter 6 we will apply it to the compliance monitoring of business processes). Our general approach also eases the unification of model use and management during different phases of the MDE process: this way, we particularly support that use and management of development models (cf. [65]) at runtime. The existing literature either is concerned with development or runtime aspects; in contrast, and following the idea from France and Rumpe [65], we argue that by making the information of development models accessible at runtime we provide richer means for analyzing and monitoring service-based systems. In the following chapter we will illustrate a scenario in which we combine different models through the power of MDE, model-aware systems can reflect on during runtime.

4.6 Summary

In this chapter we have presented model-aware services that may interact with the MORSE services for dynamic information retrieval on models and MDD projects. We have shown how traceability information is introduced and exposed. Also, we have demonstrated how services and processes can integrate with model-aware services, e.g., for monitoring purposes. For this, traceability information emitted in process events is used and dynamic model look-up is supported through MORSE that consolidates design- and runtime use and management of models. Finally, we have showcased our contributions with a case study.

CHAPTER 5

Linking Systems and Requirements through Model-Driven Engineering

A good marriage is one which allows for change and growth.
— PEARL BUCK [1]

Contents

[1] 賽珍珠

At present requirements are rarely directly connected to the systems they apply to on a conceptual level. That is, requirements as captured in form of models are not formally related to system models. Although requirement models and monitoring systems may have been developed using a model-driven engineering (MDE) methodology its system requirements are specified independently from the system models. Yet, they are not modeled in the design phase of the MDE project and linked to the system models. This is partly due to the isolation of the MDE phases as induced by the generation step: generated systems usually do not comprise traceability information and are not *per se* aware of their requirements. Another reason, also related to the isolation of the MDE phases, why requirements are expressed separately from the modeling of system models as realized in the design phase is that they may change during runtime dynamically. However, the generation step is static. Besides this it may only be partly beneficial to apply MDE for specifying system requirements and so far no incentives exist for developers to relate resulting requirements models to system models. As a consequence model-driven systems are either not aware of their requirements themselves [2] or cannot relate both system and requirement models at runtime. This, however, would ease the indexroot cause!analysisroot cause analysis in case of violations (cf. next chapter). In order to address these issues we propose to (1) utilize modeling techniques for specifying system requirements and (2) link these with the models systems are generated from. In this chapter, we investigate on the feasibility of a domain-specific requirements view in the context of a model-driven project. Finally, depicting requirements as an architectural view [97] supports separation of concerns (SoC) with regard to the various models in the system which is one of the successful approaches to manage complexity [70]. We demonstrate our approach for a compliance metadata view for process-driven service-oriented architecture (SOA) systems that specifies compliance requirements and rationales of an architecture. Thus, models are used for describing the system in terms of process models and its system requirements are specified in a requirements view model.

5.1 Compliance in Service-Oriented Architectures

We use a case from the area of business compliance in process-driven SOAs to illustrate our model-driven and domain-specific requirements view approach. Information technology (IT) compliance means in general complying with laws and regulations applying to an IT system, such as the Basel II Accord [15], the International Financial Reporting Standard [91]s (IFRSs), the Markets in Financial Instruments Directive [60] (MiFID), the Financial Security Law of France (LSF) [126], the Dutch Corporate Governance Code [169] (Code-Tabaksblat), and the

[2] I.e., additional components such as monitors non-intrusively need to examine the systems behavior in regard to its requirements.

Sarbanes-Oxley Act [49] (SOX). These cover issues such as auditor independence, corporate governance, and enhanced financial disclosure. Laws and regulations are, however, just one example of compliance concerns that occur in process-driven SOAs. There are many other rules and constraints in a SOA that have similar characteristics. Some examples are service composition and deployment rules, service execution order rules, information exchange policies, security policies, quality of service (QoS) rules, internal business rules, laws, and licenses.

Compliance concerns stemming from regulations or other compliance sources can be realized using a number of so-called *controls*. A control is any measure taken to assure a compliance requirement is met. For instance, an intrusion detection system, a firewall, or a business process realizing separation of duties (SoD) are all controls for ensuring systems security. As regulations such as SOX are not very concrete on how to realize the controls, usually, the regulations are mapped to established *norms* and *standards* describing more concretely how to realize the controls for a regulation. For example, Control Objectives for Information and Related Technology [90] (COBIT) is a standard framework that defines among others controls for ensuring system security like the examples named before. A *risk assessment* is necessary to understand for instance the possible impacts of missing or failing controls. Controls can be realized in a number of different ways, including manual controls, reports, or automated controls.

5.2 View-Based Modeling Framework

In this section, we give an overview of the View-based Modeling Framework [173, 84, 171, 172] (VbMF) that is used as a foundation for implementing our requirements view. Integrated with the Model-Aware Service Environment [82, 87, 86] (MORSE), it is a model-driven infrastructure that can generate code for process-driven SOAs in a view-based fashion. For collaborative development, all models (i.e., meta-models and their instances) are stored and versionized in the MORSE repository. These can latter on be accessed by runtime clients (i.e. by model-aware systems that we presented in the previous chapter). Changes to meta-models, when undertaken, are regularly checked for navigation compatibility (cf. Chapter 2) so that developers retrieve early feedback on the necessity of co-evolution.

After explaining the basics of VbMF, we explain the view extension mechanisms used to implement our domain-specific requirement view for compliance. Having such view extension mechanisms in place is an important part of our solution for model-driven requirements views: This way the requirements view can extend or annotate the existing models that are used to generate the system.

A typical business process in a SOA embodies various tangled concerns, such as the control flow, data processing, service and process invocations, fault handling, event handling, human

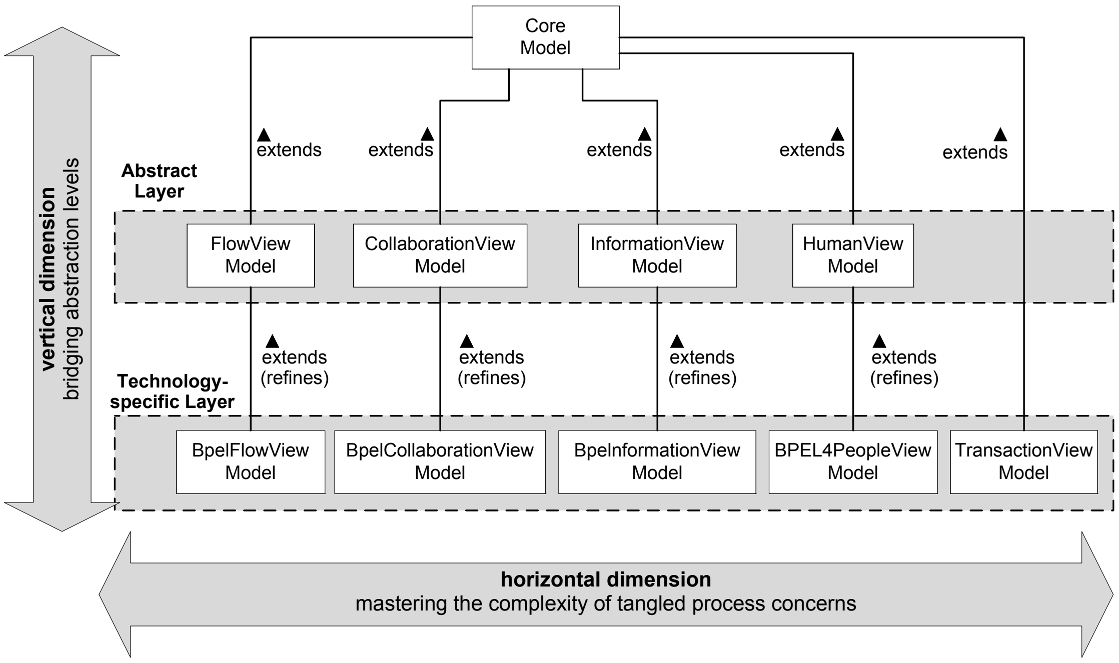


Figure 5.1: Layered Architecture of the View-based, Model-driven Approach

interactions, transactions, to name but a few. The entanglement of those concerns increases the complexity of process development and maintenance as the number of involved services and processes grow. In order to deal with this complexity, we use the notion of architectural views [97] to describe the various SOA concerns. In particular, a view is a representation of one particular concern of a process. We devise different view models for formalizing the concept of architectural view.

Figure 5.1 shows basic process concerns such as the control flow, service invocations, and data processing, in terms of the flow view, collaboration view, and information view model, respectively. In addition to these concerns, the human view (cf. [85]) captures human aspects of business processes, i.e., the participation of users in processes. Finally, the transaction view allows to define transactions within the control flow. All these view models are built up around a Core model shown in Figure 5.2. The Core model is intentionally developed for conceptually representing the essence of a business process and the services with which it interoperates. That is, the Core model covers three distinct concepts: the process, the relationships between process and the environment, i.e., the services, and the internal representation of the process, i.e., the process views. Process concerns described by the view models merely relate to these concepts in the sense that each concern involves either the process's interior or exterior, or both. In other words, the other view models derive and extend the foundational concepts provided in the Core model. As a result, the Core model plays an important role in our approach because it provides the basis for extending and integrating view models, and establishing and maintaining the dependencies between view models [173].

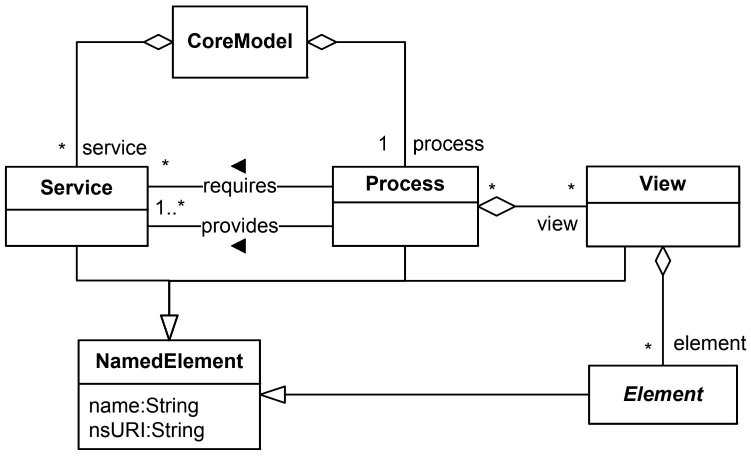


Figure 5.2: VbMF Core Model – Used for View Integration

5.2.1 Separation of Concerns

Our view-based approach is not limited to these concerns, but can be extended to cover various other concerns. For instance, human interactions, transactions, event handling have been realized as extensions [173, 85]. This view extension mechanism is also used later for introducing our domain-specific requirements view.

A new concern can be integrated into our approach by using a corresponding *New-Concern-*View model that extends the basic concepts of the Core model and defines additional concepts of that concern. By adding new view models for additional process concerns, we can extend the view-based approach along the horizontal dimension, i.e., the dimension of process concerns, to deal with the complexity caused by the various tangled process concerns.

5.2.2 Abstraction Levels

There are many stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge, skills, and needs, while IT experts merely work with low-level, technology-specific descriptions.

The model-driven development (MDD) paradigm [176] provides a potential solution to this problem by separating the platform-independent models (PIMs) and platform-specific models (PSMs). Leveraging this advantage of the MDD paradigm, we devise a model-driven stack that has two basic layers: abstract and technology-specific. The abstract layer includes the views without the technical details such that the business experts can understand and manipulate. Then, the IT experts can refine or map these abstract concepts into platform- and technology-specific views.

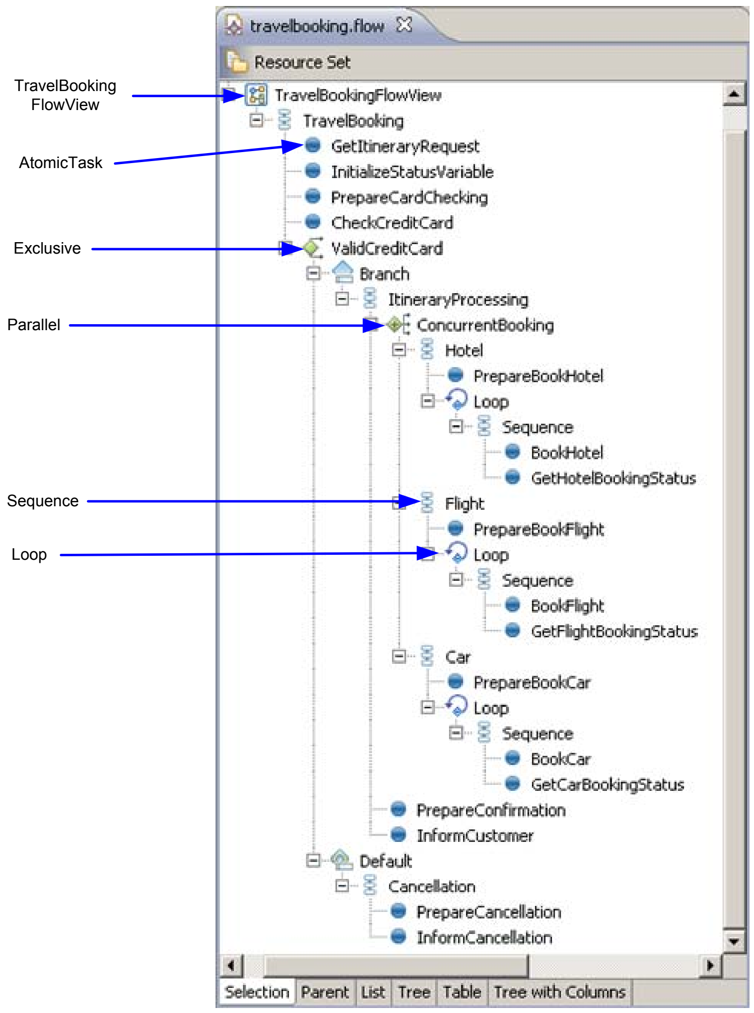
The technology-specific layer contains the views that embody concrete information of technologies or platforms. On the one hand, a technology-specific view model can be directly derived from the Core model, such as the transaction view model shown in Figure 5.1. On the other hand, a technology-specific view model can also be an extension of an abstract one, for instance, the Business Process Execution Language [139] (BPEL) collaboration view model extends the collaboration view model, the WS-BPEL Extension for People [3] (BPEL4People) view model extends the human view model, etc., by using the model refinement mechanism. By refining an abstract layer down to a technology-specific layer, our view-based approach helps bridging the abstraction levels along the vertical dimension, i.e., the dimension of abstraction, which is orthogonal to the horizontal dimension.

According to the specific needs and knowledge of the stakeholders, views can be combined to provide a richer view or a more thorough view of a certain process. For instance, IT experts may need to involve the process control flow along with service interactions which is only provided via an integration of the flow view with either the collaboration view or BPEL collaboration view.

Based on the aforementioned view model specifications, stakeholders can create different types of views for describing specific business processes. These process views can be instances of the concerns' view models, extension view models, or integrated view models (see Figure 5.1). They can be manipulated by the stakeholders to achieve a certain business goal, or adapt to new requirements in business environment or changes in technology and platform. Finally, we provide model-to-code transformations (or so-called code generations) that take these views as inputs and generate process implementations and deployment configurations. The resulting code and configurations, which may be augmented with hand-written code, can be deployed in process engines and application servers for execution.

5.2.3 Integration of Views

Views can be integrated via integration points to produce a richer view or a more thorough view of the business process. We devise a name-based matching algorithm (see Algorithm 6)



```
TravelBookingFlowView
  TravelBooking
    GetItineraryRequest
    InitializeStatusVariable
    PrepareCardChecking
    CheckCreditCard
    ValidCreditCard
      Branch
        ItineraryProcessing
          ConcurrentBooking
            Hotel
              PrepareBookHotel
              Loop
                Sequence
                  BookHotel
                  GetHotelBookingStatus
            Flight
              PrepareBookFlight
              Loop
                Sequence
                  BookFlight
                  GetFlightBookingStatus
            Car
              PrepareBookCar
              Loop
                Sequence
                  BookCar
                  GetCarBookingStatus
          PrepareConfirmation
          InformCustomer
      Default
        Cancellation
          PrepareCancellation
          InformCancellation
```

Figure 5.3: Flow View: Travel Booking Example

for realizing the view integration mechanism. This algorithm is simple, but effectively used at the view level (or model level) because from a modeler's point of view in reality, it makes sense, and is reasonable, to assign the same name to the modeling entities that pose the same functionality and semantics. Nonetheless, other view integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in our approach with reasonable effort as well (see [173] for details).

Algorithm 6: Name-Based Matching

```
  Input: Entity $v_1 \in$ View $V_1$, View $V_2$
  Output: Entity $v_2 \in V_2 \vee \emptyset$
1 begin
2     for $v_2 \in V_2$ do
3         if $v_2.name = v_1.name$ then
4             $\longleftarrow v_2$;
5     $\longleftarrow \emptyset$;
6 end
```

5.2.4 Example: Travel Booking Process

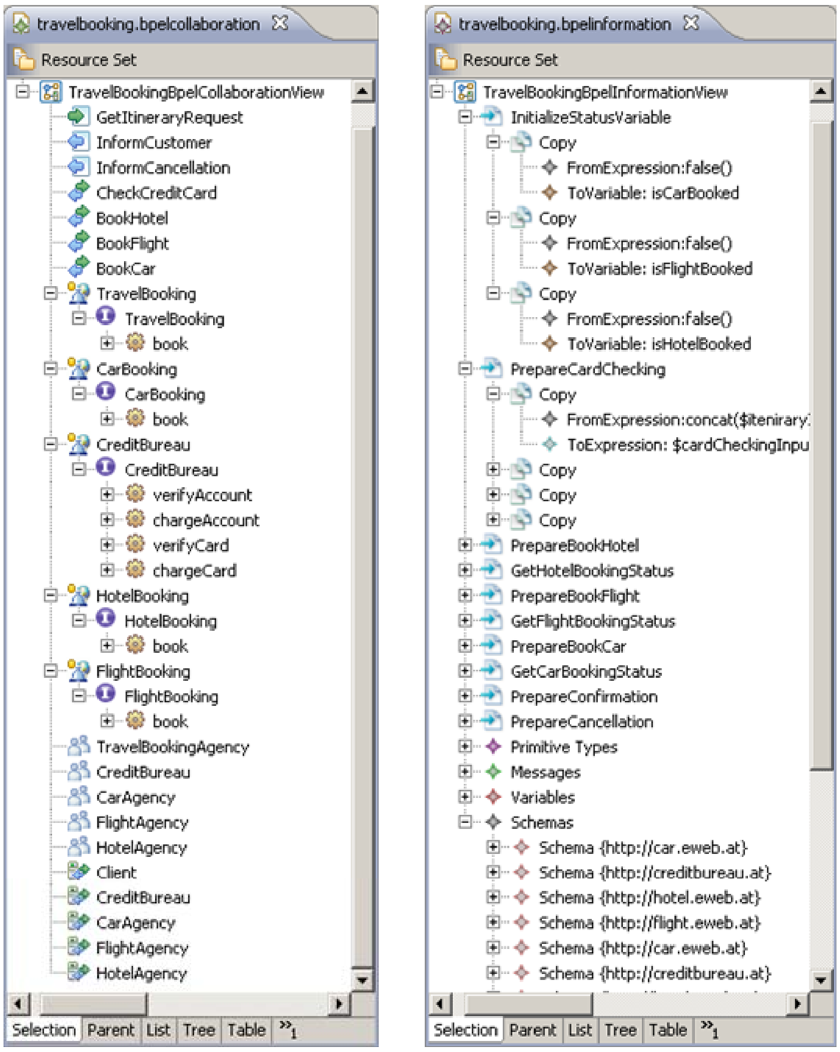
Figure 5.3 shows an example of the flow view for a travel booking application (modeled using the VbMF Eclipse perspective). As can be seen the usual concepts of process modeling can be used in the flow view. Figure 5.4 shows two extensional views in the BPEL specific variant: the BPEL collaboration and information views. The collaboration view depicts information on the components (i.e., services) the process collaborates with and how collaboration is achieved. The information view depicts information on how data is passed in, into, and out of the process, as well as the business objects the process deals with.

The views are inter–related implicitly via the integration points from the Core view. Following the name-based matching convention we use the same names – e.g., for the services – in the different views.

5.3 Design of a Domain-Specific Requirements View

In this section, we illustrate the design of a compliance metadata view as an example for a requirement view. This view allows for annotation of SOA elements with different compliance requirements. This is done by annotating process-driven model instances with the compliance metadata. This is a generic approach and can similarly be realized for requirements in other domains as well. In the case of business compliance, we elaborate on how to express compliance requirements originating from some compliance documents for process-driven SOAs using the VbMF. That is, we want to implement a compliance *control* for, e.g., a compliance regulation, standard, or norm, for a process or service.

The Compliance Metadata view provides domain-specific requirements for the domain of a process-driven SOA for compliance: It describes which parts of the SOA, i.e. which services and processes, have which roles in the compliance architecture (i.e., are they compliance controls?) and to which compliance requirements they are linked. This knowledge describes important



```
travelbooking.bpelcollaboration
Resource Set
TravelBookingBpelCollaborationView
    GetItineraryRequest
    InformCustomer
    InformCancellation
    CheckCreditCard
    BookHotel
    BookFlight
    BookCar
    TravelBooking
        TravelBooking
            book
    CarBooking
        CarBooking
            book
    CreditBureau
        CreditBureau
            verifyAccount
            chargeAccount
            verifyCard
            chargeCard
    HotelBooking
        HotelBooking
            book
    FlightBooking
        FlightBooking
            book
    TravelBookingAgency
    CreditBureau
    CarAgency
    FlightAgency
    HotelAgency
    Client
    CreditBureau
    CarAgency
    FlightAgency
    HotelAgency
Selection | Parent | List | Tree | Table | »1
```

```
travelbooking.bpelinformation
Resource Set
TravelBookingBpelInformationView
    InitializeStatusVariable
        Copy
            FromExpression:false()
            ToVariable: isCarBooked
        Copy
            FromExpression:false()
            ToVariable: isFlightBooked
        Copy
            FromExpression:false()
            ToVariable: isHotelBooked
    PrepareCardChecking
        Copy
            FromExpression:concat($itenirary
            ToExpression: $cardCheckingInpu
        Copy
        Copy
        Copy
    PrepareBookHotel
    GetHotelBookingStatus
    PrepareBookFlight
    GetFlightBookingStatus
    PrepareBookCar
    GetCarBookingStatus
    PrepareConfirmation
    PrepareCancellation
    Primitive Types
    Messages
    Variables
    Schemas
        Schema {http://car.eweb.at}
        Schema {http://creditbureau.at}
        Schema {http://hotel.eweb.at}
        Schema {http://flight.eweb.at}
        Schema {http://car.eweb.at}
        Schema {http://creditbureau.at}
Selection | Parent | List | Tree | Table | »1
```

a) TravelBooking BpelCollaborationView **b) TravelBooking BpelInformationView**

Figure 5.4: BPEL Collaboration and Information View: Travel Booking Example

architectural decisions, e.g., why certain services and processes are assembled in a certain architectural configurations. But in addition, the Compliance Metadata view has other useful aspects for the project: From it, we can automatically generate compliance documentation for offline use (i.e., Portable Document Format [93] (PDF) documents) and for online use (see also Section 5.4). Online compliance documentation is for instance used in monitoring applications that can explain the architectural configuration and rationale behind it, when a compliance violation occurs, making it easier for the operator to inspect and understand the violation.

A compliance requirement may directly relate to a process, a service, or a business con-

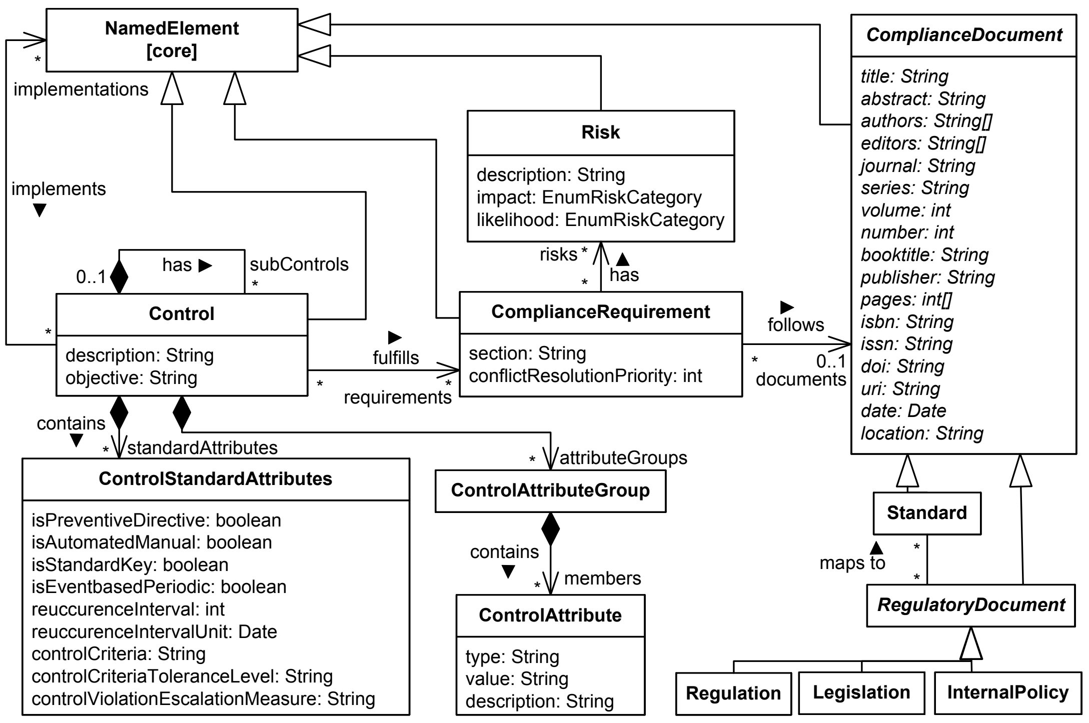


Figure 5.5: The Compliance Metadata Model

cern. Nonetheless compliance requirements not only introduce new but also depict orthogonal concerns to these: although usually related to process-driven SOA elements, they are often pervasive throughout the SOA and express independent concerns. In particular, compliance requirements can be formulated independently until applied to a SOA. As a consequence, compliance requirements can be *reused*, e.g., for different processes or process elements.

Figure 5.5 shows our proposed Compliance Metadata view. Annotation of specific SOA elements with compliance metadata is done using compliance `Controls` that relate to concrete `implementations` such as a process or service (these are defined in other views of the VbMF). A `Control` can have `subControls`. This way compliance controls can be grouped and combined. `Controls` fulfill `ComplianceRequirements` that relate to `ComplianceDocuments` such as a `Regulation`, `Legislation`, or `InternalPolicy`. Such `RegulatoryDocuments` can be mapped to `Standards` that represent another type of `ComplianceDocument`. Different categories of `ComplianceRequirements` are shown in Table 5.1.

When there exists a compliance requirement, it usually comes with risks that arise from a violation of it. `Risks` have dimensions such as `likelihood` or `impact`. In this work we provide basic support for specifying such dimensions using linear comparable constants. Of

Compliance Concern	Description
Control flow	Order and execution of process elements
Locative	Execution location of processes and process elements (e.g., a certain host, within a company, or a country)
Information	Syntax and semantics of used or produced information
Resource	Involvement of resources in processes (e.g., human resources, CPU cycles, memory, disk-space)
Temporal	Temporal constraints on process execution (e.g., deadlines, scheduled activities)

Table 5.1: Categories of Compliance Requirements

course, these can be refined with more elaborative modeling elements that allow for non-trivial functions and the use of parameters, e.g., for probability density functions.

One important aspect when specifying requirements for a system is that we want to capture their rationales (cf. pre-requirement specification traceability [72]) as well. In the context of business compliance we want to persist the relationship of a compliance requirement as derived from, e.g., a certain regulation or standard with the respective annotated SOA element. This allows for the identification and resolution of SOA elements, compliance controls, regulations, risks and compliance documents, e.g., in the case of a root cause analysis of compliance violations.

For documentation purposes and for the implementation of compliance controls the `ControlStandardAttributes` help to specify general metadata for compliance controls, e.g., if the control is automated or manual (`isAutomatedManual`). Besides these standard attributes, individual `ControlAttributes` can be defined for a compliance control within `ControlAttributeGroups`.

Figure 5.6 shows an example for compliance metadata that contains a directive from the European Union on the protection of individuals with regard to the processing of personal data. The example extends the travel booking application example presented already in Figures 5.3 and 5.4. In particular, the compliance control is implemented by the services of the travel booking process. Hence, the views in Figures 5.3 and 5.4 provide the architectural configuration of the processes and services, and Figure 5.6 provides the compliance-related rationale for the design of this configuration.

The `C1` compliance control instance for a secure transmission of personal data annotates the `TravelBooking` service of the process. The fulfilled requirement `CR1` follows the legislative document and is associated with an `AbuseRisk`. Via the name-based matching convention (see Section 5.2.3) the SOA elements are annotated in the compliance metadata view: on the left of Figure 5.6 the various services of the travel booking process are displayed next

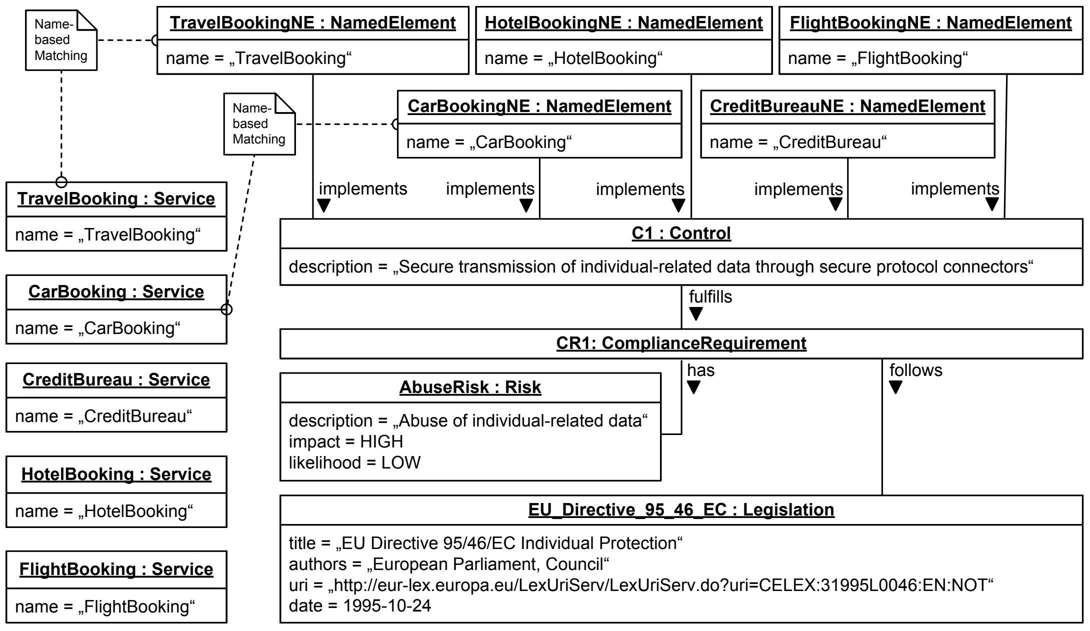

Name-based Matching
TravelBookingNE : NamedElement
name = „TravelBooking“
HotelBookingNE : NamedElement
name = „HotelBooking“
FlightBookingNE : NamedElement
name = „FlightBooking“
Name-based Matching
CarBookingNE : NamedElement
name = „CarBooking“
CreditBureauNE : NamedElement
name = „CreditBureau“
TravelBooking : Service
name = „TravelBooking“
implements
implements
implements
implements
implements
C1 : Control
description = „Secure transmission of individual-related data through secure protocol connectors“
CarBooking : Service
name = „CarBooking“
fulfills
CR1: ComplianceRequirement
CreditBureau : Service
name = „CreditBureau“
has
follows
AbuseRisk : Risk
description = „Abuse of individual-related data“
impact = HIGH
likelihood = LOW
HotelBooking : Service
name = „HotelBooking“
EU_Directive_95_46_EC : Legislation
title = „EU Directive 95/46/EC Individual Protection“
authors = „European Parliament, Council“
uri = „http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT“
date = 1995-10-24
FlightBooking : Service
name = „FlightBooking“

Figure 5.6: Compliance Metadata: Travel Booking Example

to the object instances of the compliance metadata view. The compliance control `C1` associates various `NamedElements` that hold the same name as the corresponding services from the `BpelCollaborationView` as shown in Figure 5.4).

With the proposed compliance view, it is possible to specify compliance statements such as *CR1 is a compliance requirement that follows the EU Directive 95/46/EC on Individual Protection [61] and is implemented by the TravelBooking service.* within the VbMF.

This information is useful for the project in terms of compliance documentation, and hence likely to be maintained and kept up-to-date by the developers and users of the system, because it can be used for generating the compliance documentation that is required for auditing purposes. But in this model also important architectural knowledge (AK) is captured: In particular the requirements for the process and the services that implement the control are specified. That is, this information can be used to explain the architectural configuration of the process and the services connected via a secure protocols connector.

5.4 Generating Compliance Documentation

The compliance metadata not only serves for specifying requirements of a process-driven SOA as described in the previous section but also can be used for reporting and documentation purposes. In particular, it can be used for *generating* documentations. Such documentations visualize compliance relevant information for various stakeholders, such as executive managers and auditors, and therefore, help them to quickly gain an overview of a thorough view. Hyperlinks to other documentation pages allow the user to navigate to related information or to request more specific details. In Figure 5.7 a model-to-code Xpand [168] transformation template is shown that generates HyperText Markup Language [180] (HTML) code for online documentation. The generated website displays a matrix of controls from a compliance metadata view instance that are correlated against risks.

```
<h2>Risk-Control Correlation Matrix</h2>
<table>
  <tr>
    <th>Risks/Controls</th>
«FOREACH cv.control AS c»
    <th>
      <a href="«cv.processName+
              "_C_"+c.uuid+".html"»">«c.name»</a>
    </th>
«ENDFOREACH»
  </tr>
«FOREACH cv.risk AS r»
  <tr>
    <th>
      <a href="«cv.processName+
              "_R_"+r.uuid+".html"»">«r.name»</a>
    </th>
  «FOREACH cv.control AS c»
    <td>
      «IF (c.requirements.risks.contains(r))»X«ENDIF»
    </td>
  «ENDFOREACH»
  </tr>
«ENDFOREACH»
</table>
```

Figure 5.7: A Transformation Template for Generating Compliance Documentation

Other generated documentation of the compliance metadata focuses on, e.g., the relation of compliance requirements and compliance documents, such as standards or legislative documents. Also, the *coverage* of SOA elements in regard to compliance aspects with their relation to compliance documents can be visualized and highlighted. Thus, while the generation of process code may already consider the metadata during transformation (e.g., in order to make sure that a secure protocol is used for services that are annotated accordingly), documentation that describes the requirements can be automatically created and updated. The former – i.e., the use of domain-specific requirements during code-generation, e.g., for ensuring the security of the system as in our example – is a clear incentive for a developer to specify and provide the requirements for the SOA. The latter – i.e., the model-driven generation – comes with the advantage of keeping the documentation up-to-date.

5.5 Related Work

Gotel and Finklestein [72] discuss the requirements traceability problem, i.e., introduce pre- and post- requirement specification traceability that refer to requirements production and requirements deployment. Applying a MDD approach the latter traceability is automated, e.g., for the generation of business processes and the model-aware monitoring as we will show in the next chapter. Pre-requirement specification traceability is strengthened in a requirements view that also captures rationales: in our domain-specific compliance metadata view the requirements are linked to compliance documents. Our approach is extensible and improvements can be realized by, e.g., also relating users such as compliance experts with artifacts they work on.

A recommendation for specifying software requirements [89] was published by the Institute of Electrical and Electronics Engineers (IEEE) in 1998. If desired and when appropriate, our model-driven approach can be combined with this recommendation; e.g., for generating software requirement specification (SRS) documents. For this a meta-model suitable for capturing information from these documents could be populated with information from the various views and would generate the documents similarly to how we generate the compliance documentation as shown in Section 5.4.

By annotating process-driven SOAs with requirements we establish traceability links between requirements and systems on a modeling level. Model traceability in the context of MDD has been discussed by Aizenbud et al. [5]. They propose to establish a standard traceability meta-model and point out the importance of a unique identification of artifacts across space and time. While with our transparent UUID-based model versioning approach (cf. Section 3.4.2) we satisfy the latter, in this work we have not made use of a dedicated traceability meta-model. Instead, by applying annotation via name-based matching, we establish manual relationships between requirements and systems with a predefined, implicit semantic. Yet, a traceability meta-model such as VbTrace [172] as presented for the VbMF can be introduced for this purpose, i.e., for making the requirement traceability explicit.

The requirements as captured in the compliance metadata view can also be considered as kind of AK. This is because the requirements may drive the architecture of the system (e.g., the choice to use secure connectors) and thus constitute rationals for design decisions. Thus AK that is domain-specific to the area of (compliance) requirements is recorded in our view.

In the area of AK, much work has been done in the area of architectural decision modeling.

Jansen and Bosch see software architecture as being composed of a set of design decisions [149]. They introduce a generic meta-model to capture decisions, including elements such as problems, solutions, and attributes of the AK. Another generic meta-model that is more detailed has been proposed by Zimmermann et al. [187]. Tyree and Akermann proposed a highly detailed, generic template for architectural decision capturing [174].

The related work on architectural decision modeling focuses on *generic* knowledge capturing. In contrast, our approach proposes to capture AK that is related to the system requirements in a *domain-specific* fashion, as needed by a project. Hence in our work some AK is not as explicit as in the other approaches. For example, the collaborations of components are shown in the collaboration view, whereas the other approaches rather use a typical component and connector view. The decision drivers and consequences of the decisions are reported in the compliance sources and as risks. That means, our domain-specific requirements view adopts the terminology from the compliance field, and it must be mapped to the AK terminology in order to understand the overlaps.

None of the related works provide detailed guidelines how to support the AK models or views through MDD. In contrast, this is a focus of our work.

5.6 Lessons Learned and Conclusions

We have presented a novel and generic approach of how to relate system models and system requirements models. We have demonstrated this approach for specifying requirements that are imposed on business processes within the context of compliance. Our feasibility study showed that it is feasible in a model-driven project to add an additional model-driven view that adds requirements metadata with reasonable effort.

In case of static requirements, that can be considered during code generation, the requirements view helps the model-driven approach to build conforming systems.

Hence, it makes sense to connect the requirements as captured in the requirement view with other metadata that needs to be specified in the project anyway, so that there is an additional incentive for developers. In our case, we could demonstrate an area where this is feasible: compliance in service-oriented systems. A lacking or missing compliance documentation can have severe legal consequences, which is a great incentive to specify the compliance requirements correctly. Our general approach can also be applied for custom requirements without such additional incentives.

There is the danger in our approach that only specific requirements – linked to a domain-specific area like compliance – are specified and other requirements get lost. It is the responsibility of a project to make sure that all requirements are specified.

We can conclude that it is possible and useful to add a domain-specific requirements view to a model-driven project – with reasonable effort. If extra incentives can be found, such as generating a documentation or documenting compliance, they should be used to motivate developers to keep the information in the requirements view up-to-date and consistent with the system.

Our approach assumes a model-driven approach is used for the system. It is possible to introduce our approach into a non-model-driven project (e.g., as a first step into model-driven development). For doing this at least a way to identify the existing architectural elements, such as components and connectors, must be found. But this would be considerably more work than adding the view to an existing model-driven project.

In this chapter we have presented a novel, direct linkage of system models with requirements models that we have established in the context of MDE through modeling techniques. Combined with our dynamic and reflective model-aware systems approach (presented in the previous chapter), this opens up new possibilities, e.g., for the monitoring of such systems as we will demonstrate next.

CHAPTER 6

Model-Aware Monitoring

Is it not delightful to acquire knowledge and put it into practice from time to time?

— 孔子 [1]

Contents

[1] CONFUCIUS

As service-based Internet systems get increasingly complex they become harder to manage at design time as well as at runtime. Nowadays, many systems are described in terms of precisely specified models, e.g., in the context of model-driven development (MDD). By making the information in these models accessible at runtime, we provide better means for analyzing and monitoring the service-based systems. By combing the Model-Aware Service Environment [82, 87, 86] (MORSE) approach for business processes with event-based monitoring, this chapter focuses on enabling us to monitor, interpret, and analyze the monitored information. In an industrial case study, we demonstrate how compliance monitoring can benefit from MORSE to monitor violations at runtime and how MORSE can ease the root cause analysis of such violations. Performance and scalability evaluations show the applicability of our approach for the intended use cases and that models can be retrieved during execution at low cost.

6.1 Introduction

As an illustrative case for system requirements, consider compliance to regulations: A business information system (IS) needs to comply with regulations, such as Basel II [15] or the Sarbanes-Oxley Act [49] (SOX). Runtime monitoring of service-based business processes can be used to detect violations of such regulations at execution time. If a violation is detected, a report can be generated and a root cause analysis started. In order to trace back from process instances that have caused a violation to the model elements that have driven the execution of those instances, information described in models of the system needs to be queried.

In addition, service-based systems not only require read access to the models, but also write access. In the compliance management case, for example, once a root cause analysis indicates a problem in a model, the developer should be able to (1) open the respective model for modification, (2) perform the modifications on-the-fly, and (3) add the new model version to the running system so that newly created model instances can immediately use the corrected, evolved version.

In this chapter, we propose to address these issues with MORSE. MORSE supports the tasks of the MDD life cycle by managing MDD artifacts, such as models, model instances, and transformation templates. Models are first-class citizens in MORSE, and they can be queried, changed, and updated both at design- and runtime. That is, the models in the repository can be used by the generated service-based system (e.g., the compliance monitoring infrastructure) via Web services to query and change the models while the system runs. Thanks to its generic, service-based interfaces, MORSE can be integrated in various monitoring and analysis infrastructures. We have evaluated our prototype using exhaustive performance and scalability tests to validate the applicability of the approach in practice.

6.2 Approach

Before we present some parts of the architecture in more detail, we first provide an overview of our approach. For this, we introduce model-aware and event-based monitoring. Our approach comprises the following steps:

- For the design and development of processes we apply MDD. We use the View-based Modeling Framework [173, 84, 171, 172] (VbMF) to design process models and generate Business Process Execution Language [139] (BPEL) code. We propose to store these process models in a model repository, and require that each model and model element is uniquely identifiable.
- During code generation we embed *traceability information* into the BPEL processes for relating the code with original models that we make identifiable by unique identifiers, i.e., Universally Unique Identifier [113, 98]s (UUIDs)
- The BPEL process, instrumented with traceability information, contains a BPEL extension for transmitting *low-level events*, e.g., for process invocation, containing traceability information.
- The low-level events are processed by a complex event processing (CEP) engine. *Business events* are recognized and raised.
- An interested component consumes the business events and provides for *adaptation*, *compensation*, or *synchronization*.

We illustrate our approach in Figure 6.1. Business processes, represented as process models, are at the center of our approach. We propose to store these process models (1) in a model repository that can be queried (7a and 8a). Each process model and element in the repository is identifiable by a UUID. The model repository manages and versionizes models and model instances. Additional information about related models or models in other versions can be discovered by querying the repository.

In a generation step (2), traceability information is embedded into the process definition. That is, the process and the process elements are linked in the code via UUIDs to the models. After this model-to-code transformation an executable form of a business process, such as BPEL, is provided to a process engine (3). Each process execution essentially orchestrates different business activities to realize the entire process. In order to monitor a process execution (4), we monitor the progress of each process instance through events that are emitted by the process engine (5). Within these events, we embed the UUIDs of the instance's process model. Events

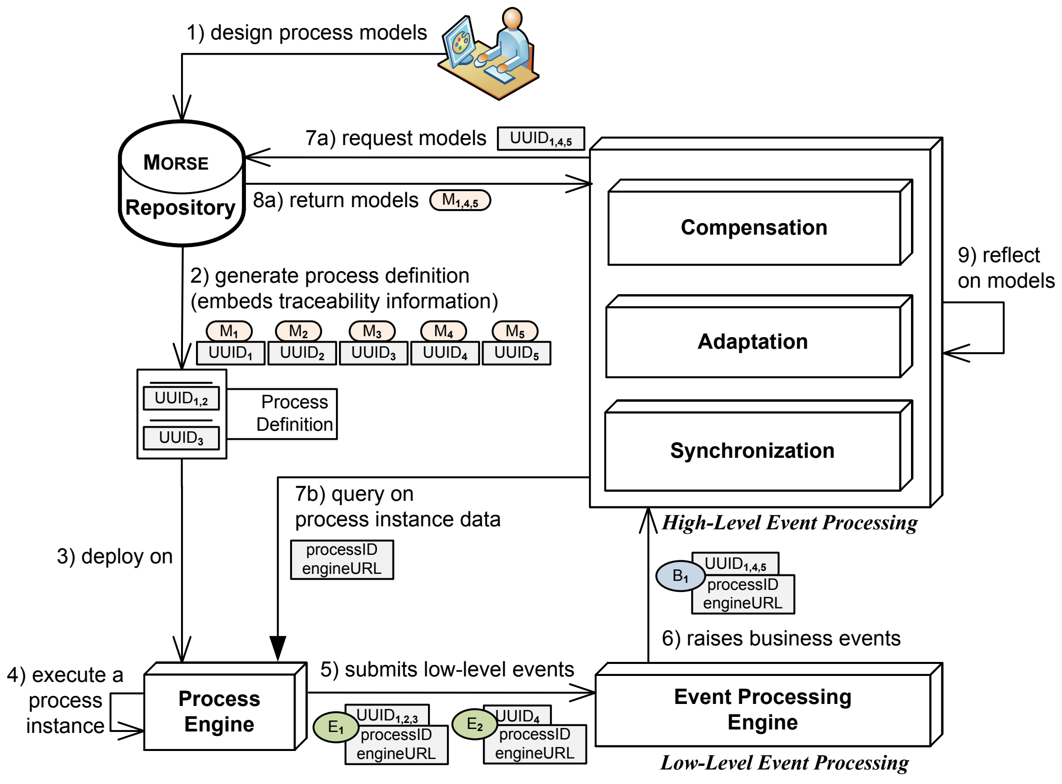

1) design process models
MORSE Repository
7a) request models
UUID1,4,5
8a) return models
M1,4,5
2) generate process definition (embeds traceability information)
M1 M2 M3 M4 M5
UUID1 UUID2 UUID3 UUID4 UUID5
UUID1,2
UUID3
Process Definition
Compensation
Adaptation
Synchronization
9) reflect on models
High-Level Event Processing
7b) query on process instance data
processID engineURL
3) deploy on
B1 UUID1,4,5 processID engineURL
6) raises business events
4) execute a process instance
Process Engine
5) submits low-level events
E1 UUID1,2,3 processID engineURL
E2 UUID4 processID engineURL
Event Processing Engine
Low-Level Event Processing

Figure 6.1: Model-Aware Monitoring Approach

emitted by the process engine are considered low-level events. In order to detect events at a business level, CEP techniques are applied (6). Business events containing, among other things, the relevant UUIDs are passed on to a component, say business intelligence component, that can perform subsequent retrieval and reflection on the process models (7a and 8a). For accessing instance data, we assume the process engine also exposes an interface for querying (7b).

In the following sections we present the details of our model-aware, event-based monitoring approach.

6.3 Case Study

To demonstrate the applicability of the presented approach we discuss a case study. In this case study, a European telecommunication company monitors the compliance of business processes. As explained in the previous chapter, compliance in an information technology (IT) context means in general complying with laws and regulations applying to an IT system (cf. [52]).

For this, a model for the compliance domain was created as displayed in Figure 6.2. It contains concepts from the compliance domain such as `Compliance Requirements` that are derived from `Compliance Sources` and which are realized by `Controls`. Compliance domain experts have developed this model together with technical experts. That is, domain experts described the concepts and their relations and technical experts specified details such as the relationships and multiplicities in iterative steps.

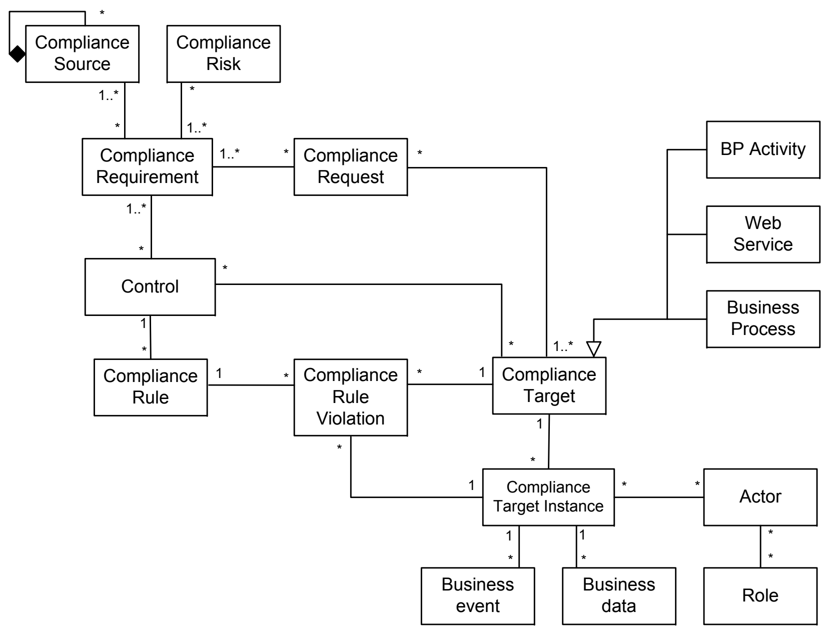


Figure 6.2: Excerpt of a Compliance Model

Note, that the resulting model from the collaboration of the stakeholders was not intended to be final but rather there was the requirement that it shall be maintained and that the system and service-oriented architecture (SOA) shall support an evolution of this model. From time to time during the case study the model was modified and a new version of the compliance model was developed. A former version of the model is shown in Figure 6.3. During the evolution of the model, some concepts have been renamed, new concepts have been introduced and relationships have been modified.

In this case study, different services in the SOA – the company operates for the execution and monitoring of its business processes – use the compliance concepts from the model. These services are realized as model-aware services. As such, they

- are able to work with and share instances of the compliance model in a distributed environment,

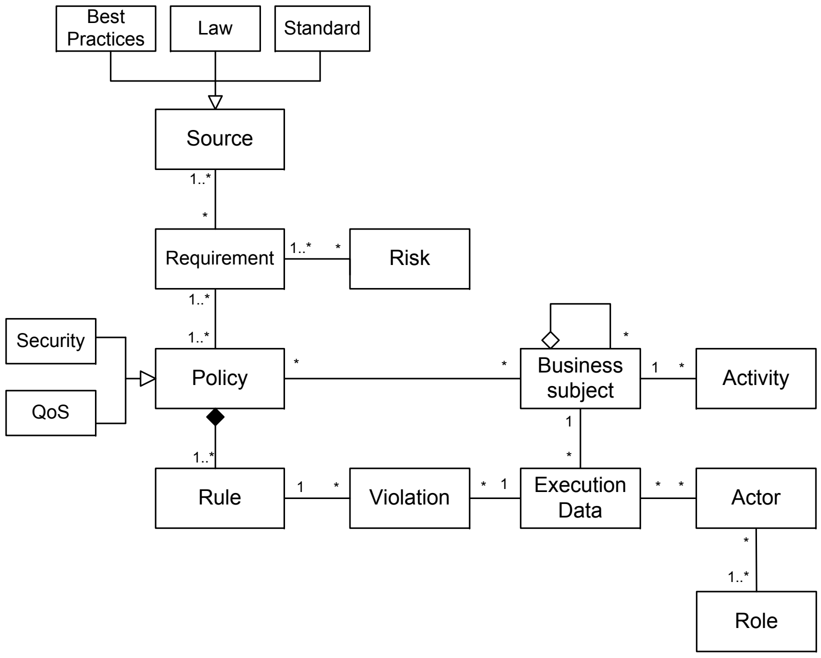


Figure 6.3: A Former Version of the Compliance Model

- can relate to and work with specific versions of compliance model instances, and
- can be upgraded to work with an evolved compliance model with a minimum of effort.

Figure 6.4 gives an overview of the relevant parts of the architecture. Business processes are executed by a process engine that emits events containing traceability information that relate to models or model elements such as a compliance requirement. Listing 6.1 shows a generated BPEL process that contains traceability information as a BPEL extension. For this the MORSE traceability matrix from Listing 4.1 is used, its XML namespace is imported and declared as a BPEL extension. The executing BPEL engine emits events for, e.g., the process activities that contain matching UUIDs. Finally, the events are processed by the monitoring infrastructure.

An online or offline analysis of the events checks the compliant process execution. The information from the monitor is used and assembled for a comprehensive reporting in a dashboard. Also notification, compensation, and adaptation tasks can be triggered by the monitor. Thus, in this setup the monitoring and adaptation in the SOA is based on the use of compliance models at runtime.

After the compliance model has been deployed (i.e., services have been generated and deployed), a compliance expert specifies reusable compliance requirements that can be used for the specification of the compliance concerns of business processes. Listing 6.2 shows an example

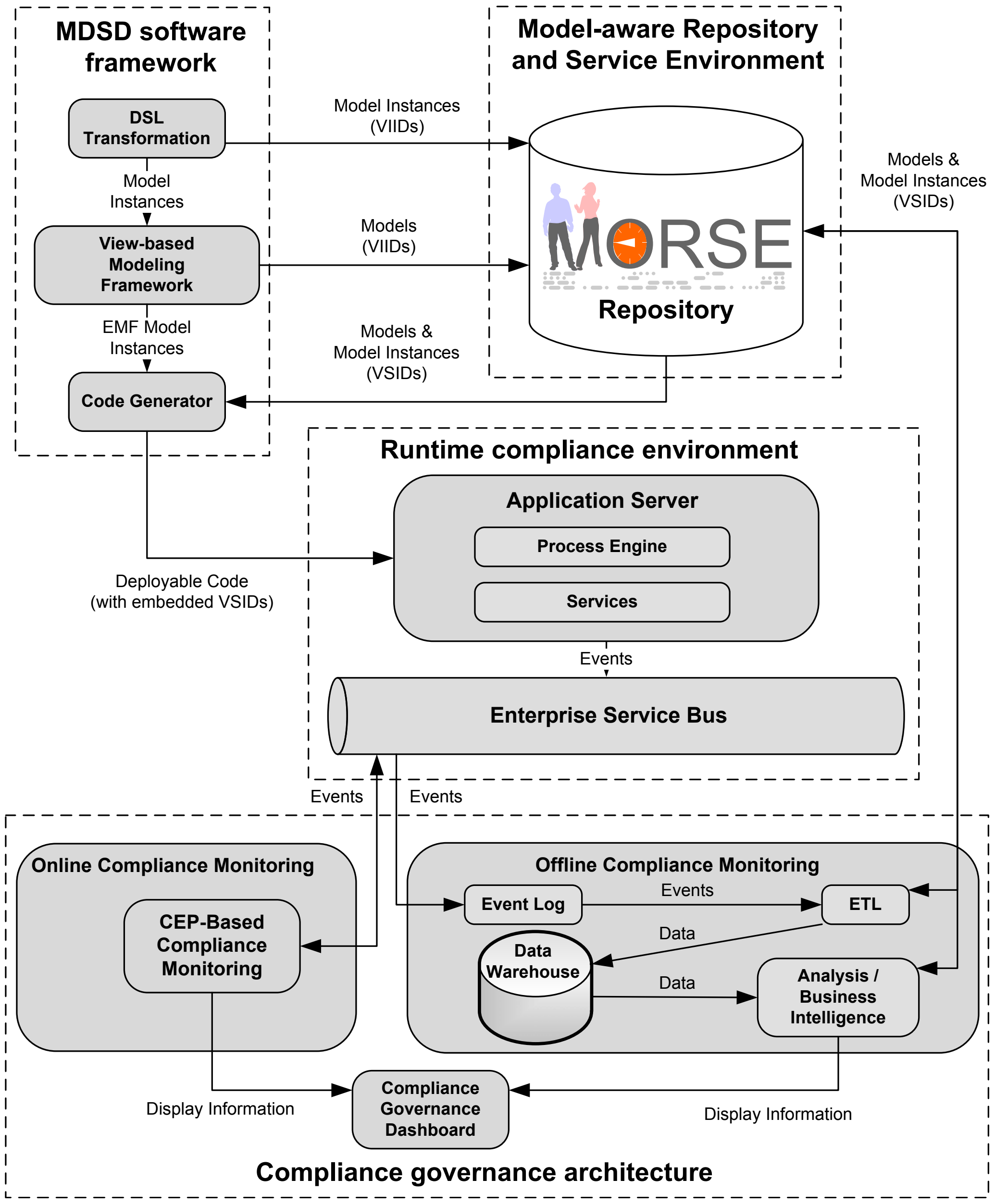

MDSD software framework
DSL Transformation
Model Instances
View-based Modeling Framework
EMF Model Instances
Code Generator
Model-aware Repository and Service Environment
Model Instances (VIIDs)
Models (VIIDs)
Models & Model Instances (VSIDs)
MORSE
Repository
Models & Model Instances (VSIDs)
Runtime compliance environment
Application Server
Process Engine
Services
Deployable Code (with embedded VSIDs)
Events
Enterprise Service Bus
Events
Events
Online Compliance Monitoring
CEP-Based Compliance Monitoring
Offline Compliance Monitoring
Event Log
Events
ETL
Data
Data Warehouse
Data
Analysis / Business Intelligence
Display Information
Compliance Governance Dashboard
Display Information
Compliance governance architecture

Figure 6.4: A Model-Aware Monitoring Architecture

from a script that populates MORSE with instances of compliance concepts using the generated service requester agents. In the example a compliance source that relates to SOX Section 409 and a compliance risk are associated with a compliance requirement. Up to Line 8 generated model code is used and the statements from Line 9 on relate to the generated code for the service

```
<process name="ReportIntrusion"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
  <extensions>
    <extension mustUnderstand="yes"
      namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
    />
  </extensions>
  <import importType="http://www.w3.org/2001/XMLSchema"
    namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
    location="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
  />
  <morse:traceability build="56810150-5bd8-4e8e-9ec5-0b88a205946b">
    <row query="/process[1]">
      <uuid>6338b114-3790-4566-a5c4-a35aa4efe41b</uuid>
      <uuid>cd2865e2-73a7-4c8d-8235-974057a40228</uuid>
      <uuid>4bcf3d70-9c23-4713-8602-3b64160c45e8</uuid>
      <uuid>c568c290-e03e-46c8-9a9a-d7afde80cc3a</uuid>
    </row>
    <row query="/process[1]/sequence[1]/receive[1]">
      <uuid>354b5161-dfab-44ef-9d52-3fb6a9d3411d</uuid>
    </row>
    <row query="/process[1]/sequence[1]/invoke[3]">
      <uuid>7d32b4f4-4f63-4223-8860-db213f7e0fe1</uuid>
    </row>
  </morse:traceability>
  <sequence>
    <!-- ... //-->
  </sequence>
</process>
```

Listing 6.1: Case Study: Traceability of the Report Intrusion Process

requester agent. In Line 9, first, a requester agent for invoking the MORSE service for compliance requirements is created. Next, the requirement object defined in Line 7 and 8 is created in MORSE as a model. Finally, in Line 11 and 12 the source and risk are created and associated with the requirement. A modeling tool realizes this interaction with the MORSE services at a technical level and assists stakeholders such as compliance experts and business administrators on a conceptual level to specify the compliance concerns of business processes at design time.

Once the compliance concerns for business processes have been specified, the processes are deployed on a process engine. During the execution of such process instances, events are emitted with data from traceability information that has been embedded into the processes and that relates to the compliance requirements. These events are analyzed by the monitor at runtime

```
CSource source = FACTORY.createCSource();
source.setDescription("SOX Sec.409");
CRisk risk = FACTORY.createCRisk();
risk.setDescription("Penalty");
risk.setImpact(EnumWeight.HIGH);
risk.setProbability(EnumWeight.LOW);
CRequirement req = FACTORY.createCRequirement();
req.setDescription("Rapid publication of Form 8−K");
CRequirementService requirementService = new CRequirementWSProxy(req);
requirementService.create();
requirementService.addSources(source);
requirementService.addRisks(risk);
```

Listing 6.2: Populating MORSE with Instances of Compliance Concepts

for the governance and adaptation. At this stage the monitor retrieves specific instances from compliance concepts such as compliance controls and related compliance rules from MORSE. Note that at runtime different versions of the compliance instances might be used in parallel. That is, stakeholders might change business processes and respective compliance concerns, yet currently running process instances must continue to operate using their original specifications in parallel. This can continue until, e.g., all old process instances have terminated. With the transparent versioning of models and the use of version-specific UUIDs at runtime the parallel execution of different process versions and compliance concerns is realized.

Finally, also the evolution of the compliance model itself is possible and supported through the automatic generation of MORSE service providers and requester agents. Model instances can be migrated through model-to-model transformation and model-aware services can be automatically instructed to work with the new model version, i.e., model code and MORSE services. Listing 6.3 shows a snipplet for setting a Maven dependency to the Web service requester agents in a model-aware service project. Thanks to Maven's transitive dependency management this is everything such a project needs to specify in order to use the necessary modules that enable the service to work with the models as well as the MORSE services at runtime. When a model evolution takes place the version number is simply updated. Please note that in case the service only operates on concepts that did not change in the evolution step, no modifications in the source code or model-driven templates that generate the model-aware service are necessary. For this, Algorithm 1 from Chapter 2 checks the navigation compatibility of a model change. If a change introduces incompatibility, however, new MORSE services will be generated and the SOA can be migrated to work with these when desired, e.g., after a review and formal approval by the management (cf. Section 2.5.5).

```
<dependency>
  <groupId>compliance.concepts</groupId>
  <artifactId>ws−proxy</artifactId>
  <version>2.0</version>
</dependency>
```

Listing 6.3: Maven Dependency for the Web Service Requester Agent

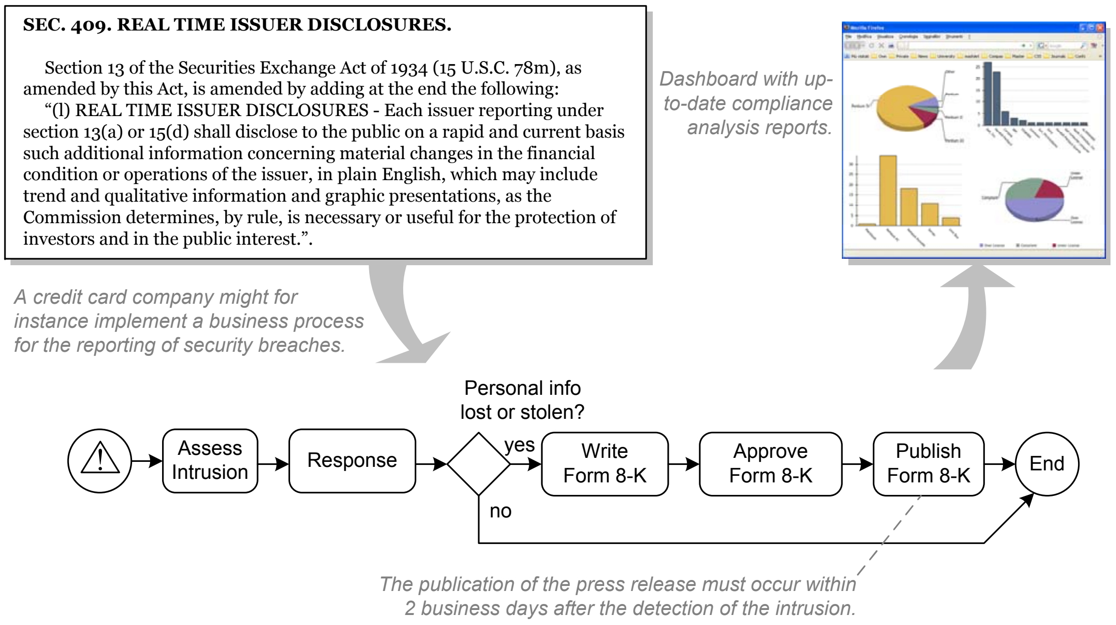
6.3.1 Business Compliance Monitoring

Let us now consider a United States (U.S.) credit card company that wants to comply with Section 409 (Real Time Issuer Disclosures) of SOX. Section 409 requires that a publicly traded company discloses information regarding material changes in the financial condition of the company in real-time (usually meaning “within two to four business days”), see also Figure 6.5. Changes in the financial condition of a company that demand for disclosure are, for example, bad debts, loss of production capacity, changes in credit ratings for the company or large clients, mergers, acquisitions, or major discoveries.

For the case of the credit card company, we consider the following reference scenario regarding the change in the financial condition: security breaches are detected in the company’s IT system, where personal information about customers might get stolen or lost. The business process in the lower part of Figure 6.5 shows a possible practice that the company may internally follow to react to a detected intrusion. After an initial assessment of the severity of the intrusion, the company immediately starts the necessary response action to mitigate risks and prevent similar future intrusions. After the response, if personal information got lost or stolen, the disclosure procedure is started. As detailed in the process, the actual disclosure is performed by filing a so-called Form 8-K report, a specific form used to notify investors and the U.S. Securities and Exchange Commission (who is in charge of controlling the compliance with SOX).

Note that full compliance with Section 409, of course, requires that all business practices in the company are compliant; the case of stolen or lost personal information represents only one out of multiple business practices in the credit card company that are subject to Section 409 of SOX.

The sole implementation of the compliant business process does not yet guarantee compliance: failures during process execution may happen (e.g., due to human errors, system failures, or the like), and the preparation and publication of the Form 8-K might be delayed, erroneous, or even forgotten. Awareness of such problems is of utmost importance to the company, in order to be able to react timely and, hence, to assure business continuity. In this regard, the ability to perform root cause analyzes to understand the reason for specific compliance violations is


SEC. 409. REAL TIME ISSUER DISCLOSURES.

Section 13 of the Securities Exchange Act of 1934 (15 U.S.C. 78m), as amended by this Act, is amended by adding at the end the following:
"(l) REAL TIME ISSUER DISCLOSURES - Each issuer reporting under section 13(a) or 15(d) shall disclose to the public on a rapid and current basis such additional information concerning material changes in the financial condition or operations of the issuer, in plain English, which may include trend and qualitative information and graphic presentations, as the Commission determines, by rule, is necessary or useful for the protection of investors and in the public interest.".

Dashboard with up-to-date compliance analysis reports.
A credit card company might for instance implement a business process for the reporting of security breaches.
Assess Intrusion
Response
Personal info lost or stolen?
yes
no
Write Form 8-K
Approve Form 8-K
Publish Form 8-K
End
The publication of the press release must occur within 2 business days after the detection of the intrusion.

Figure 6.5: SOX Example

needed.

We assume that the monitoring infrastructure in Figure 6.1 is used throughout the architecture and that MDD is used to make sure, all models are placed in the MORSE repository, and the UUIDs are used in all events emitted in the system. The MORSE repository can be used to support creating reports and running root cause analysis by making all MDD artifacts accessible at runtime. Once the cause of a violation has been understood, the developers of the company should be able to redesign the MDD artifacts (e.g., the business processes) to avoid similar violations in the future.

Figure 6.6 illustrates the interplay of MORSE, the monitoring infrastructure, and the compliance governance Web user interface (UI) when dealing with compliance violations. All models are placed in the MORSE repository. A model of a business process is annotated by compliance models. They relate to a certain regulation or specify details for an implementation. In our example, a `ComplianceRequirement` annotates the process and a `PublishDeadline` annotates the process activity `Publish Form 8-K`. These annotation models will be retrieved and evaluated by the monitoring infrastructure for detecting violations during runtime. From MORSE, the business process is automatically deployed on a business process engine (1). The business process engine emits various events such as when a process is initialized or when an

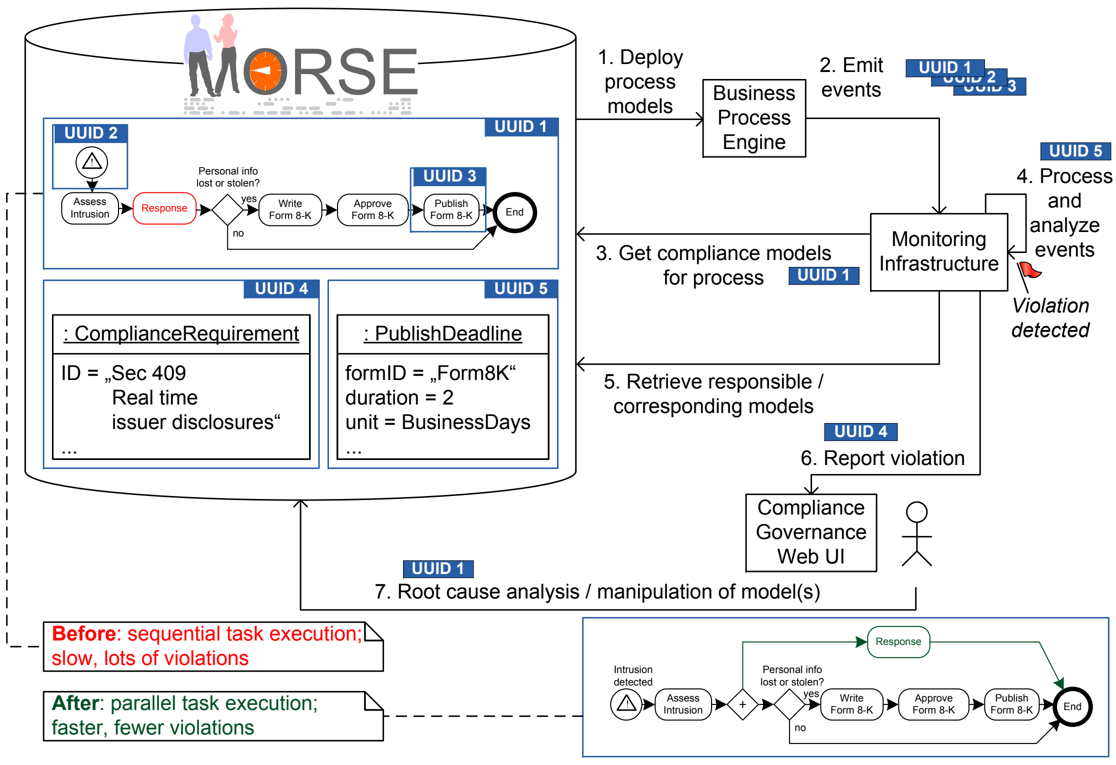

MORSE
UUID 1
UUID 2
UUID 3
UUID 4
UUID 5
Personal info lost or stolen?
Assess Intrusion
Response
yes
no
Write Form 8-K
Approve Form 8-K
Publish Form 8-K
End
: ComplianceRequirement
ID = „Sec 409 Real time issuer disclosures“
...
: PublishDeadline
formID = „Form8K“
duration = 2
unit = BusinessDays
...
1. Deploy process models
Business Process Engine
2. Emit events
4. Process and analyze events
Monitoring Infrastructure
3. Get compliance models for process
Violation detected
5. Retrieve responsible / corresponding models
6. Report violation
Compliance Governance Web UI
7. Root cause analysis / manipulation of model(s)
Before: sequential task execution; slow, lots of violations
After: parallel task execution; faster, fewer violations
Intrusion detected

Figure 6.6: Resolved SOX Example

activity is invoked or completed (2). These events contain the corresponding UUIDs and are intercepted by the monitoring infrastructure, which requests the compliance rules related to the intercepted events from MORSE (3).

Validation then takes place in online or offline operation mode (4). In case of a violation (i.e., Form 8-K has not been published within two business days according to the `PublishDeadline`), it is signaled in the dashboard. To present meaningful information to the user of the dashboard, the models responsible for the violation are retrieved from the MORSE repository and shown to the user (5). That is, the monitoring infrastructure first requests the original MDD artifact by UUID and then explores its relationships. Particularly, a compliance requirement model instance that relates to the process or process activity can be identified and displayed for adequate feedback towards the user (6).

The user can now analyze the violation by traversing the models and/or performing additional queries. That is, the dashboard or the user consults the repository for resolving and identifying the root cause of the violation. In our example, the root cause lies in the sequential structure of the control flow of the process. The user can now improve the responsible model so that new processes may not violate the compliance requirement any longer (7). In our example, the business expert improves the process so that independent tasks are executed in parallel. As a result the execution becomes faster and fewer violations occur. Using the MORSE versioning capabilities, the new model can be added to the repository, and used in MDD generations henceforth.

6.3.2 Performance and Scalability Evaluation

In the model-aware services there is only a small performance overhead for raising the events, which is often needed anyway. For instance, in our case study the events must be raised and logged for legal auditing purposes. The rest of the environment has no negative performance impact on the SOA that is monitored: For determining how many queries per second can be processed by the MORSE repository, we have conducted runtime performance and scalability tests on our prototype as shown in Figure 6.7. We measured the execution time for queries (ordinate) of a repository containing a given number of models (abscissa) and found polynomial execution time ($R^2 > 0.99$). For example, to interpret Figure 6.7, a MORSE repository with up to 2^{17} (131 072) models can process at least 15 queries ($\leq (2^{16.04}/1024/10^3)^{-1}$) within one second. This means, performance and scalability is far better than what is needed in scenarios similar to our case study that work with long-running process instances which emit events only from time to time.

Thus, models can be retrieved during execution at a low cost, especially when assuming typical Web service invocation latency. Limitations of our prototype and hardware arise with increased number of models, process instances, and events that trigger look-ups, e.g., huge MORSE repositories with more than $\approx 2^{17}$ models cannot perform 10^6 look-ups per day (arising, e.g., from 10^2 processes with 10^2 instances each that generate 10^2 events per day each). Further scalability, however, can be achieved using clusters and caches.

6.4 Related Work

There are a couple of relevant topics concerning the monitoring of SOAs such as event-based, service, or requirements monitoring. In the following we will compare our approach to related work in these areas.

6.4.1 Event-Based Monitoring

The idea of monitoring the execution of a business process is not new and has been researched previously. For example Hagen and Alonso [76] use events for direct communication between

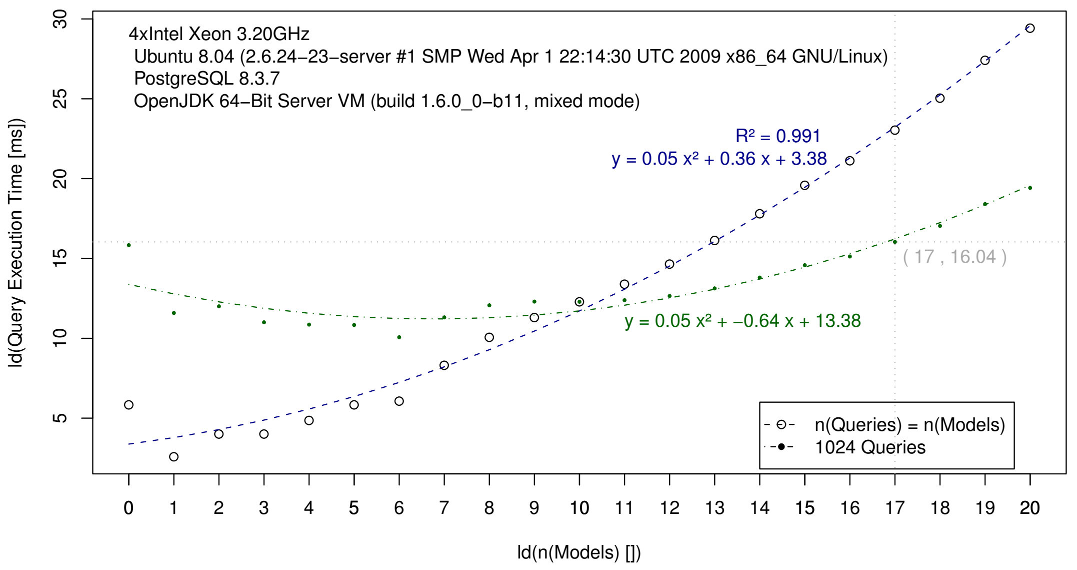


Figure 6.7: MORSE Performance and Scalability

processes. This way decisions can be made on how to proceed with the execution of process instances.

For raising events during the execution of the workflow, Casati and Discenza [36] extend the workflow model. For this the user can customize the types and occurrences of events that should be emitted. In contrast we automatically enrich the BPEL process with traceability information and – together with an implementation for a traceability BPEL extension – utilize the standard BPEL eventing as supported by BPEL engines [2].

What distinguishes our approach is to combine the eventing with a traceability to models. This allows the monitoring infrastructure to look-up and reflect on models. As a consequence the monitoring can take place at the abstraction level of the models that are – or that are related to – models as used for the design and generation of business processes. Examples are a control flow of the process or a compliance requirement that annotates a process activity as shown in Figure 6.6.

[2]cf. `http://ode.apache.org/ode-execution-events.html` for the event types supported by the Apache Orchestration Director Engine [158] (Apache ODE)

6.4.2 Service Monitoring

The topic of service (cf. [4, 48]) and process (cf. [16]) monitoring is well covered by the literature, e.g., in the areas of quality of service (QoS) and service level agreement (SLA). Yet by applying a model-aware service environment for service monitoring we contribute a novel approach, i.e., the use of service models at runtime.

A model-based design for the runtime monitoring of QoS aspects is presented by Ahluwalia et al. [4]. In particular, an interaction domain model and an infrastructure for the monitoring of deadlines are illustrated. In that approach, system functions are abstracted from interacting components. While a model-driven approach is applied for code generation, the presented model of system services is not (also) a source model for the model-driven development of the services. In contrast, MORSE manages and is aware of the service models from which systems are generated and/or relate to. This allows for supporting the monitoring and adaptation in SOAs and the root cause analysis and evolution as demonstrated in the presented case study (see Figure 6.6).

Commuzi et al. [48] explicate the link between SLA negotiation and monitoring of complex service-based systems. While being specific to SLA, their proposed monitoring architecture has similarities to our MORSE setup. We believe that by applying our approach to that work some parts of the architecture could be automatically generated while others could profit from the available MORSE service requester agents.

In contrast to these works, our MORSE approach, that allows for the management and versioning of (service) models, is a generic approach for the use of models at runtime for supporting service monitoring. It is generic because it is agnostic to the actual domain such as QoS, SLA, or compliance. In our case study we have applied our approach to compliance monitoring. The monitoring for the specific domain is realized by the model-aware services. As a consequence, all of the mentioned monitoring approaches can be supported by MORSE. For this a domain model needs to be deployed and model-aware services need to realize the domain-specific monitoring. From applying MORSE to these monitoring domains we expect profits in the areas of evolution and management.

6.4.3 Requirements Monitoring

In our approach we assume that a monitoring infrastructure is used throughout the architecture for observing and analyzing runtime events (cf. Section 6.2). While such monitoring infrastructure can be integrated with MORSE and used for the compliance checking, our work particularly focuses on relating to models from which the monitored systems have been generated. Thus, our work makes such models accessible at runtime. Note, that not only, e.g., process models but also compliance concern models are managed by MORSE. This allows for the novel and direct linkage and correlation of model-driven system and requirements models (cf. previous chapter). In this section we refer and relate to work in the areas of runtime requirements-monitoring (please also cf. Section 5.5).

Feather et al. [63] discuss an architecture and a development process for monitoring system requirements at runtime. It builds on work on goal-driven requirements engeneering (RE) [53] and runtime requirements monitoring [45].

Skene and Emmerich [150] apply MDD technologies for producing runtime requirements monitoring systems. This is, required behavior is modeled and code is generated for, e.g., the eventing infrastructure. Finally, a metadata repository collects system data and runs consistency checks to discover violations. While in our work we also showcase the generation of code for the eventing infrastructure (see Section 4.2.1), our approach assumes an existent monitoring infrastructure. In case of a violation the MORSE approach not only allows us to relate to requirement models but also to the models of the monitored system.

Chowdhary et al. [41] present a MDD framework and methodology for creating business process management (BPM) solutions. This is, a guideline is described for implementing complex BPM solutions using an MDD approach. Also, with inter alia the specification of BPM requirements and goals the framework provides runtime support for (generating) the eventing infrastructure, data warehouse, and dashboard. The presented approach allows for the monitoring and analysis of business processes in respect of their performance. Thus, similarly to our approach, compliance concerns such as QoS concerns as found in SLAs can be specified and monitored by the framework. Besides the monitoring of business processes and service-based systems in general, our approach particularly focuses on also relating to conceptual models of the systems from the runtime, not only their requirements. As a consequence, the system and end-users can directly relate to the MDD artifacts of a system in case of a violation. This allows for the subsequent reflection, adaptation, and evolution of the system. In contrast, the BPM solution supports compensation, i.e., the execution of business actions according to a decision map.

6.5 Summary

With the presented work we identified and proposed solutions for using models for service monitoring and adaptation. Monitoring and analysis of models and model elements at runtime is a real and urgent requirement in complex, Internet-based systems. Given the continuously growing adoption of MDD practices and the rising complexity of service-based systems, we have shown the usefulness of shifting the focus of model management from design time to runtime.

The MORSE approach, that treats models as first-class citizens at runtime of a service-based system, significantly eases the management of complex service-based systems by improving the analyzability, e.g., of monitoring data. For this, we proposed model-aware monitoring for the runtime monitoring of business compliance in process-driven SOAs. Our approach leverages a model repository and event-based monitoring. Business process models are stored in a model repository that can be queried. The process models are uniquely identifiable, and relations between process models can also be discovered.

We proposed a repository-based approach, in which the MDD models of a service-based system and its system requirements can be used at runtime to interactively interpret or analyze the monitored information. In the context of compliance monitoring for example and by relating processes and process activities to compliance models a business intelligence has richer means of analyzing the runtime process execution as dynamic reflection on the models and related models becomes possible. This has been demonstrated in an industrial case study for compliance management (see Section 6.3.1). The benefits of our approach can be achieved with acceptable performance and scalability impacts. While our approach facilitates monitoring, it can also be beneficial to other fields of application that profit from accessing models at runtime, e.g., in adaptive systems.

CHAPTER 7

Summary

I'm begging you, let me work!
— 手塚 治虫 [1]

In this thesis, we have presented – to the best of our knowledge – the first model repository based approach to support models at runtime in service-oriented computing (SOC). In our experiences the maintainability of models at runtime is difficult, especially in large scale systems, such as service-oriented architectures (SOAs), because the system keeps evolving during its lifetime (see, for example, the sample model in our case study in Section 6.3). Hence, different services of the SOA rely on different models or model elements in distinct versions. This problem has not yet been fully addressed in the existing literature. We have provided the transparent UUID-based model versioning approach to deal with this situation in a practice-oriented fashion. As illustrated by our case study, our approach is easily applicable, even to frequently changing central meta-models, on which many services, some developed by third parties, are based. We have used the information in the models successfully for monitoring the system and supporting some semi-automated adaptations.

However, one caveat in this approach was that initially we were only able to provide a generic interface to query the models in the repository. These are rather difficult to work with. To improve this situation, we developed XML and RESTful services (the MORSE services) that offered the specific interfaces needed for querying and traversing the models in the repository. As our approach is a model-driven approach, it was an obvious idea to use the model-driven generator for this task. However, the model-driven approach is usually used at design time; hence,

[1] OSAMU TEZUKA

we needed to extend the Model-Aware Service Environment [82, 87, 86] (MORSE) repository to automatically generate and deploy the MORSE services in the background. In this way, every time a new meta-model version is created, MORSE services are generated and deployed automatically, thus enabling users and runtime systems to easily deal with these versions. To the best of our knowledge, no other model management approach supports the automatic generation of specialized traversal and query services upon deployment of the meta-model version. This approach greatly enhances the usability of models at runtime.

The transparent versioning and the full automation of the MORSE service generation allow developers to better maintain models at runtime. To use them to better support the monitoring and adaptation in SOAs, one final problem had to be addressed: The unambiguous identification of models, model elements, and versions thereof. For this task we use UUIDs that are automatically generated into the services during the model-driven generation, making the services model-aware. This novel approach to model and model version identification allows us to generate events from services that enable monitoring and adaptation services to access the correct service model version.

Using modeling techniques to relate systems and system requirements we have shown the usefulness of our model-aware monitoring approach in the context of business compliance. To support an evolution of meta-models, on which operating model-aware services depend, we defined the notion of navigation compatible model changes, presented an algorithm that gives developers detailed feedback.

A drawback of our approach is that it combines a number of approaches and raises the overall complexity of the system. But the large degree of automation means that users usually need to interact only with the frontend parts of the MORSE for using models at runtime.

Future Work

As a first step for adopting models at runtime we have presented the MORSE approach in this thesis for the generation of model-aware services and the enabling of model-aware monitoring. We consider this work as a basis for further, constructive work and more advanced model-aware scenarios that establish models in model-aware systems as central artifacts.

To continue the idea of model *views* and also to apply it to a runtime context, it should be considered to support model-aware systems with suited, customized, e.g., context-dependent, views to models. That is, similarly to how stakeholders can be supported by the provision of views using, e.g., domain-specific languages (DSLs), model-aware systems could also profit from an appropriate presentation in form of a view model, having a specified level of abstraction. We expect such an undertaking to not only strengthen the position and role of models in software systems but also to gain experiences and useful insights in how to deal with different models, views, and abstraction levels for different stakeholders and systems.

A resulting unified framework that exceeds the basics of MORSE and supports variable view representations of models for different model-aware systems and stakeholders should also cover authorization and provenance issues in regard to models.

A question, on which we only partially focused, is how to deal with the evolution of meta-models in operating runtime model-aware systems. It would be interesting if inter-operating model-aware systems that work with different, navigation *in*compatible versions of these could collaborate (i.e., if they would dynamically reconcile the problem of changed meta-models by transforming conforming models on the fly).

It is hoped that the contributions presented in this thesis will be excelled by an advancement of models that will penetrate the fields of software engineering and service-oriented computing, that models will be assigned the role of pivotal points in service-oriented architectures, and last but not least that the pervasive adoption of models will lead us to better understand, monitor, manage, and adapt complex software systems and facilitate new forms of (model-based) collaboration.

Acknowledgments

This thesis is the result of effort and support of many, I briefly would like to thank:

Mentoring Most of all, Uwe Zdun for his scientific mentoring and guidance as well as Schahram Dustdar for providing open-minded and liberal research conditions;

Research Contributions Florian Daniel for his contribution to the SOX compliance case study [86] and Emmanuel Mulo for his work on the eventing infrastructure [83];

Development Contributions My colleague Huy Tran for his work on integrating VbMF with MORSE and the model-driven code-generation of the MORSE traceability for VbMF projects; Michael Kux for the implementation of the VDE framework [109, 50]; Wilfried Weisl for his work on the compliance metadata model and the compliance documentation generator [178]; For the technical realization of the BPEL extension for the MORSE traceability at the Apache ODE Petra Bierleutgeb as well as Steve Strauch, Oliver Kopp, and David Schumm for their work [24, 47]; Christoph Redl for his implementation of an Ajax Web application [141] for MORSE;

For stimulating discussions Amit K. Chopra, Manuel G. Clavel, Christoph Dorn, Maximilian Kögel, Fei Li, Anton Michlmayer, Ernst Oberortner, David L. Parnas, Florian Rosenberg, Hong-Linh Truong, Oktay Türetken, Martin Vasko, and Manuel Wimmer; For help in finalizing the written thesis Sylvester Keil and Benjamin R. Satzger; Last but not least my wife Helga H. Fuchs-Holmes for her encouragement and support.

Funding This thesis was financially supported by the Austrian FIT-IT research project SemBiz and the EU FP7 project COMPAS, grant no. 215175.

Bibliography

[1] *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15-19 September 2008, Munich, Germany*. IEEE Computer Society, 2008. ISBN: 978-0-7695-3373-5.

[2] Active Endpoints. *ActiveBPEL*. [accessed in December 2010]. The Apache Software Foundation, URL: `http://sf.net/projects/activebpel`. (See p. 61).

[3] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Gerhard Pfau, Karsten Plösser, Ravi Rangaswamy, Alan Rickayzen, Michael Rowley, Patrick Schmidt, Ivana Trickovic, Alex Yiu, and Matthias Zeller. *WS-BPEL Extension for People (BPEL4People), Version 1.0*. [accessed in December 2010]. June 2007. URL: `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf`. (See pp. 73, 127).

[4] Jaswinder Ahluwalia, Ingolf H. Krüger, Walter Phillips, and Michael Meisinger. “Model-based run-time monitoring of end-to-end deadlines”. In: *EMSOFT*. Ed. by Wayne Wolf. ACM, 2005, pp. 100–109. ISBN: 1-59593-091-4. (See p. 98).

[5] Neta Aizenbud-Reshef, Brian T. Nolan, Julia Rubin, and Yael Shaham-Gafni. “Model traceability”. In: *IBM Systems Journal* 45.3 (2006), pp. 515–526. (See pp. 5, 81).

[6] Marcus Alanen and Ivan Porres. “Difference and Union of Models”. In: *UML*. Ed. by Perdita Stevens, Jon Whittle, and Grady Booch. Vol. 2863. Lecture Notes in Computer Science. Springer, 2003, pp. 2–17. ISBN: 3-540-20243-9. (See pp. 30, 31).

[7] Aitor Aldazabal, Terry Baily, Felix Nanclares, Andrey Sadovykh, Christian Hein, and Tom Ritter. “Automated Model Driven Development Processes”. In: *Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration*. 2008. (See pp. 50, 52).

[8] Kerstin Altmanninger. “Models in Conflict - Towards a Semantically Enhanced Version Control System for Models”. In: *MoDELS Workshops*. Ed. by Holger Giese. Vol. 5002. Lecture Notes in Computer Science. Springer, 2007, pp. 293–304. ISBN: 978-3-540-69069-6. (See p. 50).

[9] Kerstin Altmanninger, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Wieland Schwinger, Martina Seidl, and Manuel Wimmer. “AMOR – Towards Adaptable Model Versioning”. In: *1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS ’08*. 2008. (See pp. 4, 50, 127).

[10] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. “A survey on model versioning approaches”. In: *IJWIS* 5.3 (2009), pp. 271–304. (See p. 50).

[11] Leo Apostel. “Towards the formaly study of models in the non-formal sciences”. In: *The Concept and the role of the model in mathematics and natural and social sciences*. Ed. by Hans Freudenthal. Dordrecht: D. Reidel Publishing Company, 1961, pp. 1–37. (See p. 2).

[12] AtlanMod. *Atlantic Zoo*. [accessed in December 2010]. URL: `http://emn.fr/z-info/atlanmod/index.php/Zoos`. (See pp. 26, 27, 50).

[13] Charles W. Bachman. “Data Structure Diagrams”. In: *DATA BASE* 1.2 (1969), pp. 4–10. (See p. 2).

[14] Alfred Bader. “Josef Loschmidt, the Father of Molecular Modelling”. In: *Royal Institution Proceedings* 64 (1992), pp. 197–2–05. (See p. 2).

[15] Bank for International Settlements. *Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version*. [accessed in December 2010]. June 2006. URL: `http://bis.org/publ/bcbs107.htm`. (See pp. 69, 85).

[16] Luciano Baresi, Sam Guinea, Olivier Nano, and George Spanoudakis. “Comprehensive Monitoring of BPEL Processes”. In: *IEEE Internet Computing* 14.3 (2010), pp. 50–57. (See p. 98).

[17] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*. W3C Recommendation. [accessed in December 2010]. Feb. 2004. URL: `http://w3.org/TR/owl-ref`. (See pp. 51, 129).

[18] Nelly Bencomo, Gordon S. Blair, Geoff Coulson, and Thaís Vasconcelos Batista. "Towards a Meta-Modelling Approach to Configurable Middleware". In: *RAM-SE*. Ed. by Walter Cazzola, Shigeru Chiba, Gunter Saake, and Tom Tourwé. Fakultät für Informatik, Universität Magdeburg, 2005, pp. 73–82. (See p. 66).

[19] Nelly Bencomo, Jon Whittle, Peter Sawyer, Anthony Finkelstein, and Emmanuel Letier. "Requirements reflection: requirements as runtime entities". In: *ICSE (2)*. Ed. by Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel. ACM, 2010, pp. 199–202. ISBN: 978-1-60558-719-6. (See p. 66).

[20] Avivit Bercovici, Fabiana Fournier, and Alan J. Wecker. "From Business Architecture to SOA Realization Using MDD". In: *ECMDA-FA*. Ed. by Ina Schieferdecker and Alan Hartman. Vol. 5095. Lecture Notes in Computer Science. Springer, 2008, pp. 381–392. ISBN: 978-3-540-69095-5. (See p. 35).

[21] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. *XML Path Language (XPath) 2.0*. W3C Recommendation. [accessed in December 2010]. Jan. 2007. URL: `http://w3.org/TR/xpath20/`. (See pp. 59, 131).

[22] Tim Berners-Lee, Larry Masinter, and Mark McCahill. *Uniform Resource Locators (URL)*. Request for Comments. [accessed in December 2010]. Dec. 1994. URL: `http://ietf.org/rfc/rfc1738.txt`. (See pp. 50, 130).

[23] Jean Bézivin. "On the unification power of models". In: *Software and System Modeling* 4.2 (2005), pp. 171–188. (See pp. 2, 43).

[24] Petra Bierleutgeb. *Monitoring of a Model-Aware SOA*. Bachelor Thesis. Vienna, Austria: Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Sept. 2009. (See p. 104).

[25] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. [accessed in December 2010]. Oct. 2004. URL: `http://w3.org/TR/xmlschema-2/`. (See p. 59).

[26] Gordon S. Blair, Nelly Bencomo, and Robert B. France. "Models@ run.time". In: *IEEE Computer* 42.10 (2009), pp. 22–27. (See p. 66).

[27] Grady Booch and Alan W. Brown. "Collaborative Development Environments". In: *Advances in Computers* 59 (2003), pp. 1 –27. ISSN: 0065-2458. DOI: `10.1016/S0065-2458(03)59001-5`. (See p. 38).

[28] Artur Boronat, José A. Carsí, Isidro Ramos, and Patricio Letelier. “Formal Model Merging Applied to Class Diagram Integration”. In: *Electr. Notes Theor. Comput. Sci.* 166 (2007), pp. 5–26. (See p. 30).

[29] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.1.* [accessed in December 2010]. Aug. 2006. URL: `http://w3.org/TR/xml11/`. (See pp. 49, 59, 131).

[30] Petra Brosch, Philip Langer, Martina Seidl, and Manuel Wimmer. “Towards end-user adaptable model versioning: The By-Example Operation Recorder”. In: *CVSM ’09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 55–60. ISBN: 978-1-4244-3714-6. DOI: `10.1109/CVSM.2009.5071723`. (See p. 50).

[31] Petra Brosch, Martina Seidl, Konrad Wieland, Manuel Wimmer, and Philip Langer. “We can work it out: Collaborative Conflict Resolution in Model Versioning”. In: *ECSCW 2009: Proceedings of the 11th European Conference on Computer Supported Cooperative Work*. Springer, 2009, pp. 207–214. DOI: `10.1007/978-1-84882-854-4_12`. (See pp. 4, 50).

[32] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. “A manifesto for model merging”. In: *GaMMa ’06: Proceedings of the 2006 international workshop on Global integrated model management*. Shanghai, China: ACM, 2006, pp. 5–12. ISBN: 1-59593-410-3. DOI: `10.1145/1138304.1138307`. (See p. 30).

[33] Alex Buckley. *A Metadata Facility for the Java^{TM} Programming Language*. Java Specification Request 175. Java Community Process, Sept. 2004. URL: `http://jcp.org/en/jsr/detail?id=175`. (See p. 60).

[34] Frank Budinsky, Ed Merks, and David Steinberg. *Eclipse Modeling Framework 2.0 (2nd Edition)*. Addison-Wesley Professional, 2006. ISBN: 0321331885. (See p. 39).

[35] Canonical Ltd. *Launchpad*. [accessed in December 2010]. URL: `http://launchpad.net`. (See p. 38).

[36] Fabio Casati and Angela Discenza. “Supporting workflow cooperation within and across organizations”. In: *Proceedings of the 2000 ACM symposium on Applied computing*. Como, Italy: ACM, 2000, pp. 196–202. ISBN: 1-58113-240-9. DOI: `10.1145/335603.335742`. (See p. 97).

[37] Peter P. Chen. “The Enity-Relationship Model: Toward a Unified View of Data”. In: *VLDB*. Ed. by Douglas S. Kerr. ACM, 1975, p. 173. (See p. 2).

[38] Betty H. C. Cheng, Peter Sawyer, Nelly Bencomo, and Jon Whittle. “A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty”. In: *MoDELS*. Ed. by Andy Schürr and Bran Selic. Vol. 5795. Lecture Notes in Computer Science. Springer, 2009, pp. 468–483. ISBN: 978-3-642-04424-3. (See pp. 4, 7).

[39] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0: Part 2: Adjuncts*. W3C Recommendation. [accessed in December 2010]. June 2007. URL: `http://w3.org/TR/wsdl20-adjuncts`. (See pp. 62, 129).

[40] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation. [accessed in December 2010]. June 2007. URL: `http://w3.org/TR/wsdl20`. (See pp. 35, 58, 130).

[41] Pawan Chowdhary, Kumar Bhaskaran, Nathan S. Caswell, Henry Chang, Tian Chao, Shyh-Kwei Chen, Michael J. Dikun, Hui Lei, Jun-Jang Jeng, Shubir Kapoor, Christian A. Lang, George A. Mihaila, Ioana Stanoi, and Liangzhao Zeng. “Model Driven Development for Business Performance Management”. In: *IBM Systems Journal* 45.3 (2006), pp. 587–606. (See pp. 6, 99).

[42] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. “Automating Co-evolution in Model-Driven Engineering”. In: *EDOC*. IEEE Computer Society, 2008, pp. 222–231. ISBN: 978-0-7695-3373-5. (See p. 31).

[43] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Commun. ACM* 13.6 (1970), pp. 377–387. (See p. 2).

[44] Codehaus. *Codehaus*. [accessed in December 2010]. URL: `http://codehaus.org`. (See p. 38).

[45] Don Cohen, Martin S. Feather, K. Narayanaswamy, and Stephen S. Fickas. “Automatic monitoring of software requirements”. In: *ICSE ’97: Proceedings of the 19th international conference on Software engineering*. Boston, Massachusetts, United States: ACM, 1997, pp. 602–603. ISBN: 0-89791-914-9. DOI: `10.1145/253228.253493`. (See p. 99).

[46] CollabNet Corp. *CollabNet*. [accessed in December 2010]. URL: `http://collab.net`. (See p. 38).

[47] COMPAS Consortium. *Supporting Infrastructure – Process Engine, Process Artefact Repository, Process Generation Tool*. FP7-215175 Deliverable 4.4. [accessed in December 2010]. 2009. URL: `http://compas-ict.eu/D4.4.pdf`. (See p. 104).

[48] Marco Comuzzi, Constantinos Kotsokalis, George Spanoudakis, and Ramin Yahyapour. "Establishing and Monitoring SLAs in Complex Service Based Systems". In: *ICWS*. IEEE, 2009, pp. 783–790. (See pp. 7, 98).

[49] Congress of the United States. *Public Company Accounting Reform and Investor Protection Act (Sarbanes-Oxley Act), Pub.L. 107-204, 116 Stat. 745*. [accessed in December 2010]. July 2002. URL: `http://gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html`. (See pp. 70, 85, 130).

[50] SemBiz Consortium. *Prototype Implementation*. FIT-IT SemBiz Deliverable 2.3. [accessed in December 2010]. May 2008. URL: `http://sembiz.at/attach/D2.3.pdf`. (See p. 104).

[51] Chessman Corrêa, Leonardo Gresta Paulino Murta, and Cláudia Maria Lima Werner. "Odyssey-MEC: Model Evolution Control in the Context of Model-Driven Architecture". In: *SEKE*. Knowledge Systems Institute Graduate School, 2008, pp. 67–72. ISBN: 1-891706-22-5. (See p. 52).

[52] Florian Daniel, Fabio Casati, Vincenzo D'Andrea, Emmanuel Mulo, Uwe Zdun, Schahram Dustdar, Steve Strauch, David Schumm, Frank Leymann, Samir Sebahi, Fabien De Marchi, and Mohand-Said Hacid. "Business Compliance Governance in Service-Oriented Architectures". In: *AINA*. Ed. by Irfan Awan, Muhammad Younas, Takahiro Hara, and Arjan Durresi. IEEE Computer Society, 2009, pp. 113–120. ISBN: 978-0-7695-3638-5. (See p. 87).

[53] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. "Goal-Directed Requirements Acquisition". In: *Sci. Comput. Program.* 20.1-2 (1993), pp. 3–50. (See p. 99).

[54] Linda DeMichiel. *Java^{TM} Persistence 2.0*. Java Specification Request 317. Java Community Process, Dec. 2009. URL: `http://jcp.org/en/jsr/detail?id=317`. (See pp. 43, 49, 128).

[55] Dick Grune. *Concurrent Versions System*. [accessed in December 2010]. Free Software Foundation, June 1986. URL: `http://nongnu.org/cvs`. (See pp. 39, 41, 127).

[56] Mark Doliner. *Cobutera*. [accessed in December 2010]. URL: `http://cobertura.sf.net`. (See p. 47).

[57] Paul Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk.* Addison-Wesley Professional, 2007. ISBN: 9780321336385. (See p. 39).

[58] Alexander Egyed. “Automated abstraction of class diagrams”. In: *ACM Trans. Softw. Eng. Methodol.* 11.4 (2002), pp. 449–491. (See pp. 17, 18, 31).

[59] Alexander Egyed. “Compositional and Relational Reasoning during Class Abstraction”. In: *UML*. Ed. by Perdita Stevens, Jon Whittle, and Grady Booch. Vol. 2863. Lecture Notes in Computer Science. Springer, 2003, pp. 121–137. ISBN: 3-540-20243-9. (See pp. 17, 31).

[60] European Parliament and Council. *Directive 2004/39/EC on markets in financial instruments*. [accessed in December 2010]. Apr. 2004. URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:02004L0039-20060428:EN:NOT`. (See pp. 69, 129).

[61] European Parliament and Council. *Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. [accessed in December 2010]. Oct. 1995. URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT`. (See p. 79).

[62] Jean-Marie Favre. “Meta-Model and Model Co-evolution within the 3D Software Space”. In: *International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA), in conjunction with ICSM 2003*. 2003. (See p. 6).

[63] Martin S. Feather, Stephen S. Fickas, Axel van Lamsweerde, and Christophe Ponsard. “Reconciling system requirements and runtime behavior”. In: *Software Specification and Design, 1998. Proceedings. Ninth International Workshop on*. Apr. 1998, pp. 50–59. DOI: `10.1109/IWSSD.1998.667919`. (See p. 99).

[64] Robert B. France, James M. Bieman, and Betty H. C. Cheng. “Repository for Model Driven Development (ReMoDD)”. In: *MoDELS Workshops*. Ed. by Thomas Kühne. Vol. 4364. Lecture Notes in Computer Science. Springer, 2006, pp. 311–317. ISBN: 978-3-540-69488-5. (See p. 38).

[65] Robert B. France and Bernhard Rumpe. “Model-driven Development of Complex Software: A Research Roadmap”. In: *FOSE*. Ed. by Lionel C. Briand and Alexander L. Wolf. 2007, pp. 37–54. (See pp. 66, 67).

[66] Roman Frigg and Stephan Hartmann. “Models in Science”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2009. [accessed in December 2010]. 2009. URL: `http://plato.stanford.edu/archives/spr2009/entries/models-science/`. (See p. 2).

[67] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, Nov. 1994. ISBN: 0201633612. (See p. 61).

[68] Erich Gamma, Richard Helm, Ralph E. Johnson, and John M. Vlissides. “Design Patterns: Abstraction and Reuse of Object-Oriented Design”. In: *ECOOP*. Ed. by Oscar Nierstrasz. Vol. 707. Lecture Notes in Computer Science. Springer, 1993, pp. 406–431. ISBN: 3-540-57120-5. (See p. 61).

[69] GForgeGroup. *GForge*. [accessed in December 2010]. URL: `http://gforge.org`. (See p. 38).

[70] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991. (See pp. 3, 69).

[71] Allan Gibbard and Hal R. Varian. “Economic models”. In: *Journal of Philosophy* 75.11 (1978), pp. 664–677. (See p. 2).

[72] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. “An analysis of the requirements traceability problem”. In: *Requirements Engineering, 1994., Proceedings of the First International Conference on*. Apr. 1994, pp. 94 –101. DOI: `10.1109/ICRE.1994.292398`. (See pp. 78, 81).

[73] Boris Gruschko, Dimitrios S. Kolovos, and Richard F. Paige. “Towards Synchronizing Models with Evolving Metamodels”. In: *Workshop on Model-Driven Software Evolution at CSMR 2007*. 2007. (See p. 31).

[74] Marc Hadley. *Web Application Description Language*. W3C Submission. [accessed in December 2010]. Aug. 2009. URL: `http://w3.org/Submission/wadl`. (See pp. 58, 130).

[75] Marc Hadley and Paul Sandoz. *JAX-RS: The Java^{TM} API for RESTful Web Services*. Java Specification Request 311. Java Community Process, Oct. 2008. URL: `http://jcp.org/en/jsr/detail?id=311`. (See pp. 48, 128).

[76] Claus Hagen and Gustavo Alonso. “Beyond the black box: event-based inter-process communication in process support systems”. In: *19th IEEE International Conference on Distributed Computing Systems*. 1999, pp. 450–457. DOI: `10.1109/ICDCS.1999.776547`. (See p. 96).

[77] Tony Hansen, Ted Hardie, and Larry Masinter. *Guidelines and Registration Procedures for New URI Schemes*. Request for Comments. [accessed in December 2010]. Feb. 2006. URL: `http://ietf.org/rfc/rfc4395.txt`. (See pp. 44, 50, 51, 58, 130).

[78] Stefan Haustein and Jörg Pleumann. "A model-driven runtime environment for Web applications". In: *Software and System Modeling* 4.4 (2005), pp. 443–458. (See p. 38).

[79] Christian Hein, Tom Ritter, and Michael Wagner. "Model-Driven Tool Integration with ModelBus". In: *Workshop Future Trends of Model-Driven Development*. 2009. (See pp. 50, 52).

[80] Carsten Hentrich and Uwe Zdun. "Patterns for Process-Oriented Integration in Service-Oriented Architectures". In: *EuroPLoP*. Ed. by Uwe Zdun and Lise B. Hvatum. UVK - Universitätsverlag Konstanz, 2006, pp. 141–198. ISBN: 978-3-87940-813-9. (See p. 6).

[81] Markus Herrmannsdörfer, Sebastian Benz, and Elmar Jürgens. "COPE - Automating Coupled Evolution of Metamodels and Models". In: *ECOOP*. Ed. by Sophia Drossopoulou. Vol. 5653. Lecture Notes in Computer Science. Springer, 2009, pp. 52–76. ISBN: 978-3-642-03012-3. (See pp. 29, 31).

[82] Ta'id Holmes. *Model-Aware Service Environment (*MORSE*)*. [accessed in December 2010]. Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Sept. 2008. URL: `http://www.infosys.tuwien.ac.at/prototype/morse`. (See pp. 8, 25, 27, 35, 36, 53, 56, 70, 85, 102, 129).

[83] Ta'id Holmes, Emmanuel Mulo, Uwe Zdun, and Schahram Dustdar. "Model-Aware Monitoring of SOAs for Compliance". In: ed. by Li Fei and Schahram Dustdar. Springer, 2011, pp. 117–136. ISBN: 978-3-7091-0414-9. DOI: `10.1007/978-3-7091-0415-6`. (See p. 104).

[84] Ta'id Holmes, Huy Tran, Uwe Zdun, and Schahram Dustdar. *A Service-Oriented Toolchain for Model-Driven, View-Based Business Process Design and Deployment*. Poster and Presentation, 4th European Conference on Model Driven Architecture - Foundations and Applications, Tools and Services Track. June 2008. (See pp. 61, 70, 86, 130).

[85] Ta'id Holmes, Huy Tran, Uwe Zdun, and Schahram Dustdar. "Modeling Human Aspects of Business Processes – A View-Based, Model-Driven Approach". In: *ECMDA-FA*. Ed. by Ina Schieferdecker and Alan Hartman. Vol. 5095. Lecture Notes in Computer Science. Springer, 2008, pp. 246–261. ISBN: 978-3-540-69095-5. DOI: `10.1007/978-3-540-69100-6_17`. (See pp. 71, 72).

[86] Ta'id Holmes, Uwe Zdun, Florian Daniel, and Schahram Dustdar. "Monitoring and Analyzing Service-Based Internet Systems through a Model-Aware Service Environment". In: *CAiSE*. Ed. by Barbara Pernici. Vol. 6051. Lecture Notes in Computer Science. Springer, June 2010, pp. 98–112. ISBN: 978-3-642-13093-9. DOI: 10.1007/978-3-642-13094-6_9. (See pp. 8, 25, 35, 36, 53, 56, 70, 85, 102, 104, 129).

[87] Ta'id Holmes, Uwe Zdun, and Schahram Dustdar. "MORSE: A Model-Aware Service Environment". In: *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC)*. Ed. by Markus Kirchberg, Patrick C. K. Hung, Barbara Carminati, Chi-Hung Chi, Rajaraman Kanagasabai, Emanuele Della Valle, Kun-Chan Lan, and Ling-Jyh Chen. IEEE, Dec. 2009, pp. 470–477. ISBN: 978-1-4244-5336-8. DOI: 10.1109/APSCC.2009.5394083. (See pp. 8, 25, 35, 36, 39, 53, 56, 70, 85, 102, 129).

[88] "IEEE Guide to Software Configuration Management". In: *ANSI/IEEE Std 1042-1987* (1988). DOI: 10.1109/IEEESTD.1988.94582. (See pp. 43, 129).

[89] "IEEE Recommended Practice for Software Requirements Specifications". In: *IEEE Std 830-1998* (1998), p. i. DOI: 10.1109/IEEESTD.1998.88286. (See p. 81).

[90] Information Systems Audit and Control Association. *Control Objectives for Information and Related Technology (CobiT)*. [accessed in December 2010]. 1996. URL: http://www.isaca.org/cobit. (See pp. 70, 127).

[91] International Accounting Standards Committee (IASC) Foundation. *International Financial Reporting Standards*. [accessed in December 2010]. URL: http://www.ifrs.org/IFRSs. (See pp. 69, 128).

[92] International Organization for Standardization. *ISO 10007:2003 Quality management systems – Guidelines for configuration management*. [accessed in December 2010]. Mar. 2003. URL: http://iso.org/iso/catalogue_detail.htm?csnumber=36644. (See pp. 43, 129).

[93] International Organization for Standardization. *ISO 32000-1:2008 Document management – Portable document format – Part 1: PDF 1.7*. [accessed in December 2010]. July 2008. URL: http://iso.org/iso/catalogue_detail.htm?csnumber=51502. (See pp. 76, 129).

[94] International Organization for Standardization. *ISO/IEC 19501:2005 Information technology – Open Distributed Processing – Unified Modeling Language (UML), v1.4.2*. [accessed in December 2010]. Apr. 2005. URL: http://omg.org/cgi-bin/doc?formal/05-04-01. (See p. 3).

[95] International Organization for Standardization. *ISO/IEC 19503:2005 Information technology – XML Metadata Interchange (XMI)*. [accessed in December 2010]. Nov. 2005. URL: `http://iso.org/iso/catalogue_detail.htm?csnumber=32622`. (See pp. 39, 131).

[96] International Organization for Standardization. *ISO/IEC 19505-1 Information technology – OMG Unified Modeling Language (OMG UML) Version 2.1.2 – Part 1: Infrastructure*. [accessed in December 2010]. Nov. 2007. URL: `http://iso.org/iso/catalogue_detail.htm?csnumber=32624`. (See p. 3).

[97] International Organization for Standardization. *ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description of software-intensive systems*. [accessed in December 2010]. Sept. 2007. URL: `http://iso.org/iso/catalogue_detail.htm?csnumber=45991`. (See pp. 3, 69, 71).

[98] International Telecommunication Union. *ISO/IEC 9834-8 Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components*. [accessed in December 2010]. Sept. 2004. URL: `http://itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf`. (See pp. 9, 36, 57, 86, 126, 130).

[99] Frédéric Jouault and Jean Bézivin. "KM3: A DSL for Metamodel Specification". In: *FMOODS*. Ed. by Roberto Gorrieri and Heike Wehrheim. Vol. 4037. Lecture Notes in Computer Science. Springer, 2006, pp. 171–185. ISBN: 3-540-34893-X. (See pp. 51, 128).

[100] Kohsuke Kawaguchi. *Hudson*. [accessed in December 2010]. URL: `http://hudson-ci.org`. (See p. 47).

[101] Kim Letkeman. *Comparing and merging UML models in IBM Rational Software Architect*. [accessed in May 2010]. July 2005. URL: `http://www.ibm.com/developerworks/rational/library/05/712_comp/`. (See p. 30).

[102] Dimitrios S. Kolovos, Richard F. Paige, and Fiona Polack. "Merging Models with the Epsilon Merging Language (EML)". In: *MoDELS*. Ed. by Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio. Vol. 4199. Lecture Notes in Computer Science. Springer, 2006, pp. 215–229. ISBN: 3-540-45772-0. (See p. 30).

[103] Fabio Kon, Fábio M. Costa, Gordon S. Blair, and Roy H. Campbell. "The case for reflective middleware". In: *Commun. ACM* 45.6 (2002), pp. 33–38. (See p. 66).

[104] Jitendra Kotamraju. *The Java^{TM} API for XML-Based Web Services (JAX-WS) 2.0.* Java Specification Request 224. Java Community Process, Apr. 2006. URL: `http://jcp.org/en/jsr/detail?id=224`. (See pp. 48, 128).

[105] Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel, eds. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. ACM, 2010. ISBN: 978-1-60558-719-6.

[106] Gerhard Kramler, Gerti Kappel, Thomas Reiter, Elisabeth Kapsammer, Werner Retschitzegger, and Wieland Schwinger. "Towards a semantic infrastructure supporting model-based tool integration". In: *GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management*. Shanghai, China: ACM, 2006, pp. 43–46. ISBN: 1-59593-410-3. DOI: `10.1145/1138304.1138314`. (See p. 50).

[107] Philippe Kruchten. "The 4+1 View Model of Architecture". In: *IEEE Software* 12.6 (1995), pp. 42–50. (See p. 3).

[108] John A. Kunze and Thomas Baker. *The Dublin Core Metadata Element Set*. Request for Comments. [accessed in December 2010]. Aug. 2007. URL: `http://ietf.org/rfc/rfc5013.txt`. (See pp. 40, 127).

[109] Michael Kux. *Validation/Deployment/Execution (VDE) Framework*. Practical. Vienna, Austria: Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Nov. 2007. (See pp. 61, 104).

[110] Maximilian Kögel and Jonas Helming. "EMFStore: a model repository for EMF models". In: *ICSE (2)*. Ed. by Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel. ACM, 2010, pp. 307–308. ISBN: 978-1-60558-719-6. (See pp. 50, 51).

[111] Maximilian Kögel, Markus Herrmannsdörfer, Yang Li, Jonas Helming, and Joern David. "Comparing State- and Operation-Based Change Tracking on Models". In: *Enterprise Distributed Object Computing Conference, IEEE International* 0 (2010), pp. 163–172. ISSN: 1541-7719. DOI: `10.1109/EDOC.2010.15`. (See pp. 50, 51).

[112] Maximilian Kögel, Markus Herrmannsdörfer, Otto von Wesendonk, and Jonas Helming. "Operation-based conflict detection". In: *Proceedings of the 1st International Workshop on Model Comparison in Practice*. IWMCP '10. Malaga, Spain: ACM, 2010, pp. 21–30. ISBN: 978-1-60558-960-2. DOI: `10.1145/1826147.1826154`. (See pp. 50, 51).

[113] Paul J. Leach, Michael Mealling, and Rich Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. Request for Comments. [accessed in December 2010]. July 2005. URL: `http://ietf.org/rfc/rfc4122.txt`. (See pp. 9, 36, 43, 44, 57, 86, 126, 130).

[114] Tihamer Levendovszky, Daniel Balasubramanian, Anantha Narayanan, and Gabor Karsai. “A Novel Approach to Semi-automated Evolution of DSML Model Transformation”. In: *SLE*. Ed. by Mark van den Brand, Dragan Gasevic, and Jeff Gray. Vol. 5969. Lecture Notes in Computer Science. Springer, 2009, pp. 23–41. ISBN: 978-3-642-12106-7. (See p. 32).

[115] Linus Torvalds. *Git*. [accessed in December 2010]. Apr. 2005. URL: `http://git-scm.com`. (See pp. 39, 41).

[116] JFrog Ltd. *Artifactory*. [accessed in December 2010]. JFrog Ltd., URL: `http://artifactory.sf.net`. (See p. 48).

[117] Roberto Lucchi and Manuel Mazzara. “A pi-calculus based semantics for WS-BPEL”. In: *J. Log. Algebr. Program.* 70.1 (2007), pp. 96–118. (See p. 61).

[118] Daniel Lucrédio, Renata Pontin de Mattos Fortes, and Jon Whittle. “MOOGLE: A Model Search Engine”. In: *MoDELS*. Ed. by Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter. Vol. 5301. Lecture Notes in Computer Science. Springer, 2008, pp. 296–310. ISBN: 978-3-540-87874-2. (See p. 52).

[119] Martin Pool. *Bazaar*. [accessed in December 2010]. Canonical Ltd., Mar. 2005. URL: `http://bazaar.canonical.com`. (See pp. 39, 41).

[120] Larry Masinter and Karen Sollins. *Functional Requirements for Uniform Resource Names*. Request for Comments. [accessed in December 2010]. Dec. 1994. URL: `http://ietf.org/rfc/rfc1737.txt`. (See pp. 44, 130).

[121] Matt Mackall. *Mercurial*. [accessed in December 2010]. Apr. 2005. URL: `http://mercurial.selenic.com`. (See pp. 39, 41).

[122] Martin Matula. *NetBeans Metadata Repository*. [accessed in July 2009]. NetBeans Community, URL: `http://mdr.netbeans.org`. (See pp. 50, 51, 129).

[123] Philip Mayer, Andreas Schröder, and Nora Koch. “MDD4SOA: Model-Driven Service Orchestration”. In: *EDOC*. IEEE Computer Society, 2008, pp. 203–212. ISBN: 978-0-7695-3373-5. (See pp. 6, 35).

[124] Manuel Mazzara and Ivan Lanese. “Towards a Unifying Theory for Web Services Composition”. In: *WS-FM*. Ed. by Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro. Vol. 4184. Lecture Notes in Computer Science. Springer, 2006, pp. 257–272. ISBN: 3-540-38862-1. (See p. 61).

[125] Marjan Mernik, Jan Heering, and Anthony M. Sloane. “When and how to develop domain-specific languages”. In: *ACM Comput. Surv.* 37.4 (2005), pp. 316–344. (See p. 3).

[126] Ministre de l’économie, des finances et de l’industrie. *loi de sécurité financière*. [accessed in December 2010]. Feb. 2003. URL: `http://senat.fr/leg/pjl02-166.html`. (See pp. 69, 128).

[127] Leonardo Murta, Chessman Corrêa, João Gustavo Prudêncio, and Cláudia Werner. “Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System”. In: *CVSM ’08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*. Leipzig, Germany: ACM, 2008, pp. 25–30. ISBN: 978-1-60558-045-6. DOI: `10.1145/1370152.1370159`. (See pp. 50, 52).

[128] Leonardo Gresta Paulino Murta, Hamilton L. R. Oliveira, Cristine R. Dantas, Luiz Gustavo Lopes, and Cláudia Maria Lima Werner. “Odyssey-SCM: An integrated software configuration management infrastructure for UML models”. In: *Sci. Comput. Program.* 65.3 (2007), pp. 249–274. (See pp. 50, 52).

[129] Ernst Oberortner, Uwe Zdun, and Schahram Dustdar. “Domain-Specific Languages for Service-Oriented Architectures: An Explorative Study”. In: *ServiceWave*. Ed. by Petri Mähönen, Klaus Pohl, and Thierry Priol. Vol. 5377. Lecture Notes in Computer Science. Springer, 2008, pp. 159–170. ISBN: 978-3-540-89896-2. (See p. 3).

[130] Object Management Group, Inc. *MDA Guide Version 1.0.1*. [accessed in December 2010]. June 2003. URL: `http://omg.org/cgi-bin/doc?omg/03-06-01`. (See pp. 3, 13, 128).

[131] Object Management Group, Inc. *Meta-Object Facility*. [accessed in December 2010]. Apr. 2002. URL: `http://omg.org/mof`. (See pp. 3, 15, 39, 42, 48, 125, 128, 129).

[132] Object Management Group, Inc. *Meta-Object Facility 1.4*. [accessed in December 2010]. Apr. 2002. URL: `http://omg.org/spec/MOF/1.4`. (See pp. 51, 52).

[133] Object Management Group, Inc. *Object Constraint Language (OCL)*. [accessed in December 2010]. May 2006. URL: `http://omg.org/spec/OCL`. (See pp. 14, 129).

[134] Object Management Group, Inc. *Unified Modeling Language (UML)*. [accessed in December 2010]. Mar. 2000. URL: `http://omg.org/spec/UML`. (See pp. 2, 15, 39, 130).

[135] Object Management Group, Inc. *XML Metadata Interchange (XMI®)*. [accessed in December 2010]. URL: `http://omg.org/spec/XMI`. (See pp. 39, 49, 131).

[136] Dirk Ohst, Michael Welle, and Udo Kelter. "Differences between versions of UML diagrams". In: *ESEC / SIGSOFT FSE*. ACM, 2003, pp. 227–236. (See pp. 30, 31).

[137] Hamilton L. R. Oliveira, Leonardo Gresta Paulino Murta, and Cláudia Maria Lima Werner. "Odyssey-VCS: a flexible version control system for UML model elements". In: *SCM*. ACM, 2005, pp. 1–16. ISBN: 1-59593-310-7. (See pp. 43, 50, 52, 130).

[138] openArchitectureWare.org. *openArchitectureWare*. [accessed in December 2010]. openArchitectureWare.org, URL: `http://openarchitectureware.org`. (See pp. 40, 129).

[139] Organization for the Advancement of Structured Information Standards. *Web Service Business Process Execution Language Version 2.0*. OASIS Standard. [accessed in July 2010]. OASIS Web Services Business Process Execution Language (WSBPEL) TC, Jan. 2007. URL: `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`. (See pp. 59, 73, 86, 127).

[140] PostgreSQL Global Development Group. *PostgreSQL*. [accessed in December 2010]. PostgreSQL Global Development Group, 1997. URL: `http://postgresql.org`. (See p. 43).

[141] Christoph Redl. *Browsing and Managing Domain Models with a Generic, Resource-Oriented Ajax Web Application*. Bachelor Thesis. Vienna, Austria: Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Apr. 2010. (See p. 104).

[142] James E. Rumbaugh, Michael R. Blaha, William J. Premerlani, Frederick Eddy, and William E. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991. ISBN: 0-13-630054-5. (See p. 2).

[143] Ernest Rutherford. "The scattering of alpha and beta particles by matter and the structure of the atom". In: *Philosophical Magazine* 21 (1911). [accessed in December 2010], pp. 669–688. URL: `http://www.math.ubc.ca/~cass/rutherford/rutherford.html`. (See p. 2).

[144] Ina Schieferdecker and Alan Hartman, eds. *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*. Vol. 5095. Lecture Notes in Computer Science. Springer, 2008. ISBN: 978-3-540-69095-5.

[145] Petri Selonen. “A Review of UML Model Comparison Techniques”. In: *Proceedings of the 5th Nordic Workshop on Model Driven Engineering*. Ronneby, Sweden, Aug. 2007, pp. 37–51. (See p. 30).

[146] Michael E. Senko, Edward B. Altman, Morton M. Astrahan, and P. L. Fehder. “Data Structures and Accessing in Data-Base Systems. I: Evolution of Information Systems”. In: *IBM Systems Journal* 12.1 (1973), pp. 30–44. (See p. 2).

[147] Michael E. Senko, Edward B. Altman, Morton M. Astrahan, and P. L. Fehder. “Data Structures and Accessing in Data-Base Systems. II: Information Organization”. In: *IBM Systems Journal* 12.1 (1973), pp. 45–63. (See p. 2).

[148] Michael E. Senko, Edward B. Altman, Morton M. Astrahan, and P. L. Fehder. “Data Structures and Accessing in Data-Base Systems. III: Data Representations and the Data Independent Accessing Model”. In: *IBM Systems Journal* 12.1 (1973), pp. 64–93. (See p. 2).

[149] *Sixth Working IEEE / IFIP Conference on Software Architecture (WICSA 2007), 6-9 January 2005, Mumbai, Maharashtra, India*. IEEE Computer Society, 2007. ISBN: 978-0-7695-2744-4. (See p. 82).

[150] James Skene and Wolfgang Emmerich. “Engineering Runtime Requirements-Monitoring Systems Using MDA Technologies”. In: *TGC*. Ed. by Rocco De Nicola and Davide Sangiorgi. Vol. 3705. Lecture Notes in Computer Science. Springer, 2005, pp. 319–333. ISBN: 3-540-30007-4. (See pp. 6, 99).

[151] Prawee Sriplakich, Xavier Blanc, and Marie-Pierre Gervais. “Supporting transparent model update in distributed CASE tool integration”. In: *SAC*. Ed. by Hisham Haddad. ACM, 2006, pp. 1759–1766. ISBN: 1-59593-108-2. (See pp. 50, 52).

[152] Herbert Stachowiak. *Allgemeine Modelltheorie*. Wien [u.a.]: Springer, 1973, p. 494. ISBN: 3-211-81106-0. (See p. 2).

[153] Sebastian Stein, Stefan Kühne, Jens Drawehn, Sven Feja, and Werner Rotzoll. “Evaluation of OrViA Framework for Model-Driven SOA Implementations: An Industrial Case Study”. In: *BPM*. Ed. by Marlon Dumas, Manfred Reichert, and Ming-Chien Shan. Vol. 5240. Lecture Notes in Computer Science. Springer, 2008, pp. 310–325. ISBN: 978-3-540-85757-0. (See p. 35).

[154] Perdita Stevens, Jon Whittle, and Grady Booch, eds. *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings*. Vol. 2863. Lecture Notes in Computer Science. Springer, 2003. ISBN: 3-540-20243-9.

[155] The Apache Software Foundation. *Apache Continuum*. [accessed in December 2010]. The Apache Software Foundation, URL: `http://continuum.apache.org`. (See p. 47).

[156] The Apache Software Foundation. *Apache Derby*. [accessed in December 2010]. The Apache Software Foundation, 1997. URL: `http://db.apache.org/derby`. (See p. 43).

[157] The Apache Software Foundation. *Apache Maven*. [accessed in December 2010]. The Apache Software Foundation, URL: `http://maven.apache.org`. (See pp. 47, 48).

[158] The Apache Software Foundation. *Apache ODE*. [accessed in December 2010]. The Apache Software Foundation, 2007. URL: `http://ode.apache.org`. (See pp. 61, 97, 127).

[159] The Apache Software Foundation. *Apache Subversion*. [accessed in December 2010]. The Apache Software Foundation, 2000. URL: `http://subversion.apache.org`. (See pp. 39, 41, 130).

[160] The Eclipse Foundation. *Connected Data Objects (CDO) Model Repository*. [accessed in December 2010]. The Eclipse Foundation, 2005. URL: `http://wiki.eclipse.org/CDO`. (See pp. 50, 51, 127).

[161] The Eclipse Foundation. *Eclipse*. [accessed in December 2010]. The Eclipse Foundation, Nov. 2001. URL: `http://eclipse.org`. (See p. 125).

[162] The Eclipse Foundation. *Eclipse Modeling Framework Project (EMF)*. [accessed in December 2010]. The Eclipse Foundation, 2002. URL: `http://eclipse.org/modeling/emf`. (See pp. 27, 31, 32, 39, 42, 48, 125, 128).

[163] The Eclipse Foundation. *Eclipse Persistence Services Project (EclipseLink)*. [accessed in December 2010]. The Eclipse Foundation, 2008. URL: `http://eclipse.org/eclipselink`. (See p. 43).

[164] The Eclipse Foundation. *EMF Compare*. [accessed in December 2010]. The Eclipse Foundation, Oct. 2006. URL: `http://wiki.eclipse.org/EMF_Compare`. (See p. 30).

[165] The Eclipse Foundation. *KM3*. [accessed in December 2010]. The Eclipse Foundation, URL: `http://wiki.eclipse.org/KM3`. (See pp. 51, 128).

[166] The Eclipse Foundation. *Net4j*. [accessed in December 2010]. The Eclipse Foundation, URL: `http://wiki.eclipse.org/Net4j`. (See p. 51).

[167] The Eclipse Foundation. *Teneo*. [accessed in December 2010]. The Eclipse Foundation, 2005. URL: `http://wiki.eclipse.org/Teneo`. (See pp. 43, 51).

[168] The Eclipse Foundation. *Xpand*. [accessed in December 2010]. The Eclipse Foundation, URL: `http://wiki.eclipse.org/Xpand`. (See p. 80).

[169] The Netherlands Corporate Governance Committee. *The Dutch corporate governance code*. [accessed in December 2010]. Dec. 2003. URL: `http://commissiecorporategovernance.nl/page/downloads/CODEDEFENGELSCOMPLEETII.pdf`. (See pp. 69, 127).

[170] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. [accessed in December 2010]. Oct. 2004. URL: `http://w3.org/TR/xmlschema-1/`. (See p. 59).

[171] Huy Tran, Ta'id Holmes, Uwe Zdun, and Schahram Dustdar. "Modeling Process-Driven SOAs – a View-Based Approach". In: ed. by Jorge Cardoso and Wil van der Aalst. Handbook of Research on Business Process Modeling. Information Science Reference, Apr. 2009. Chap. 2, pp. 27–48. ISBN: 978-1-60566-288-6. DOI: `10.4018/978-1-60566-288-6`. URL: `http://www.igi-global.com/reference/details.asp?ID=33287`. (See pp. 70, 86, 130).

[172] Huy Tran, Uwe Zdun, and Schahram Dustdar. "VbTrace: using view-based and model-driven development to support traceability in process-driven SOAs". In: *Software and Systems Modeling* (2009), pp. 1–25. ISSN: 1619-1366. DOI: `10.1007/s10270-009-0137-0`. (See pp. 70, 81, 86, 130).

[173] Huy Tran, Uwe Zdun, and Schahram Dustdar. "View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA". In: *BPSC*. Ed. by Witold Abramowicz and Leszek A. Maciaszek. Vol. 116. LNI. GI, 2007, pp. 105–124. ISBN: 978-3-88579-210-9. (See pp. 27, 70, 72, 74, 86, 130).

[174] Jeff Tyree and Art Akerman. "Architecture Decisions: Demystifying Architecture". In: *IEEE Software* 22.2 (2005), pp. 19–27. (See p. 82).

[175] Axel Uhl. "Model-Driven Development in the Enterprise". In: *IEEE Software* 25.1 (2008), pp. 46–49. (See p. 4).

[176] Markus Völter and Thomas Stahl. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006. ISBN: 9780470025703. (See pp. 3, 73).

[177] Guido Wachsmuth. "Metamodel Adaptation and Model Co-adaptation". In: *ECOOP*. Ed. by Erik Ernst. Vol. 4609. Lecture Notes in Computer Science. Springer, 2007, pp. 600–624. ISBN: 978-3-540-73588-5. (See p. 32).

[178] Wilfried Weisl. *Konzepte für die Dokumentation von Compliance mittels modell-getriebener Software Entwicklung und SOA*. Master Thesis. Vienna, Austria: Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Nov. 2008. (See p. 104).

[179] Dave Winer. *RSS 2.0 Specification*. [accessed in December 2010]. RSS Advisory Board, Mar. 1999. URL: `http://rssboard.org/rss-specification`. (See pp. 66, 129).

[180] *XHTML^{TM} 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation. [accessed in December 2010]. Aug. 2002. URL: `http://w3.org/TR/xhtml1/`. (See pp. 80, 128).

[181] Qian Yang, J. Jenny Li, and David Weiss. "A survey of coverage based testing tools". In: *AST '06: Proceedings of the 2006 international workshop on Automation of software test*. Shanghai, China: ACM, 2006, pp. 99–103. ISBN: 1-59593-408-1. DOI: `10.1145/1138929.1138949`. (See p. 47).

[182] Yanping Yang, QingPing Tan, and Yong Xiao. "Verifying web services composition based on hierarchical colored petri nets". In: *IHIS*. Ed. by Axel Hahn, Sven Abels, and Liane Haak. ACM, 2005, pp. 47–54. (See p. 61).

[183] Uwe Zdun. "A DSL toolkit for deferring architectural decisions in DSL-based software design". In: *Information & Software Technology* 52.7 (2010), pp. 733–748. (See pp. 22, 125).

[184] Uwe Zdun. *Frag*. [accessed in December 2010]. URL: `http://frag.sf.net`. (See pp. 22, 125).

[185] Uwe Zdun. "Tailorable language for behavioral composition and configuration of software components". In: *Computer Languages, Systems & Structures* 32.1 (2006), pp. 56–82. (See p. 47).

[186] Uwe Zdun, Carsten Hentrich, and Schahram Dustdar. "Modeling process-driven and service-oriented architectures using patterns and pattern primitives". In: *TWEB* 1.3 (2007). (See p. 6).

[187] Olaf Zimmermann, Uwe Zdun, Thomas Gschwind, and Frank Leymann. “Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method”. In: *WICSA*. IEEE Computer Society, 2008, pp. 157–166. (See p. 82).

Glossary

COMPAS	A European Union (EU) Seventh Framework Programme (FP7) research project (FP7-ICT-2007-1.1.2 Service and Software Architectures, Infrastructures and Engineering) with grant number 215175. It uses model-driven techniques, domain-specific languages, and service-oriented infrastructure software to enable organizations developing business compliance solutions easier and faster. 104
Eclipse	The Eclipse integrated development environment (IDE) [161] is an open-source multi-language software development environment. 51
Ecore	The Eclipse Modeling Framework [162] (EMF) implementation of the Essential Meta Object Facility [131] (EMOF) M3 meta-meta-model (cf. [131]). 39, 42, 47–49, 51
FIT-IT	Research, Innovation, Technology – Information Technology (FIT-IT) is an Austrian research program that focuses on high-quality research in the area of information and communication technology. 104, 126
Frag	A dynamic programming language [184, 183]. 47
model-aware monitoring	Runtime monitoring that takes place at the abstraction level of models. ii, 8–10, 56, 81, 100, 102

model-aware service	A service that consumes, produces, or reflects on models. ii, 4, 5, 9–11, 35–38, 41, 47–49, 54, 56–59, 61, 63, 67, 88, 92, 96, 98, 102
model-aware system	A system that consumes, produces, or reflects on models. ii, 8, 36, 38, 54, 66, 67, 70, 83, 102, 103
navigation compatible model	A model that, when compared to another model, contains the navigability of that model. 8
navigation compatible model change	A change in a model that does not break any navigability of the original model. 8–10, 102
SemBiz	An Austrian FIT-IT research project (2005-1 Semantic Systems). It aims at bridging the gap between the business level perspective and the technical implementation level in business process management (BPM) by semantic descriptions of business processes along with respective tool support. 104
transparent UUID-based model versioning	A versioning technique for Universally Unique Identifier [113, 98] (UUID)-based model repositories that hides the complexity of implicit versioning of models and model elements from users while respecting the UUID principle. 9, 10, 44, 48, 53, 81, 101
Universally Unique Identifier	UUID is a standard for unique identifiers in (distributed) software system development. ii, 37
version-independent object	An object that is subject to change. 44, 46, 126
version-independent UUID	A UUID that identifies a version-independent object. 44
version-specific object	An object that is not subject to change. 44–46, 126
version-specific UUID	A UUID that identifies a version-specific object. 44

Acronyms

Ajax	Asynchronous JavaScript and XML. 104
AK	architectural knowledge. 79, 81, 82
AMOR	Adaptable Model Versioning [9]. 50, 51
Apache ODE	Apache Orchestration Director Engine [158]. 61, 97, 104
BPEL	Business Process Execution Language [139]. 59–63, 73, 75, 86, 89, 97, 104
BPEL4People	WS-BPEL Extension for People [3]. 73
BPM	business process management. 99, 126
CDE	collaborative development environment. 35, 38, 39, 53, 54
CDO	Connected Data Objects model repository [160]. 50, 51
CEP	complex event processing. 66, 67, 86, 87
CI	continuous integration. 39, 47
COBIT	Control Objectives for Information and Related Technology [90]. 70
Code-Tabaksblat	Dutch Corporate Governance Code [169]. 69
CRUD	create, retrieve, update, and delete. 49
CVS	Concurrent Versions System [55]. 41
DC	Dublin Core [108]. 40
DSL	domain-specific language. 4, 46, 102
EDA	event-driven architecture. 9

EMF	Eclipse Modeling Framework [162]. 27, 30, 32, 39, 43, 48–52, 54, 125
EMOF	Essential Meta Object Facility [131]. 39, 42, 48, 125
ERD	entity-relationship diagram. 2
ERM	entity-relationship model. 2
EU	European Union. 104, 125
FIT-IT	Research, Innovation, Technology – Information Technology. 125
FP7	Seventh Framework Programme. 104, 125
HTML	HyperText Markup Language [180]. 80
ID	identifier. 43, 50–52
IDE	integrated development environment. 28, 51, 125
IEEE	Institute of Electrical and Electronics Engineers. 81
IFRS	International Financial Reporting Standard [91]. 69
IR	information retrieval. 4, 5, 37, 39, 41, 54, 57, 67
IS	information system. 85
ISO	International Organization for Standardization. 3
IT	information technology. 69, 72, 73, 87, 93
JAX-RS	Java API for RESTful Web Services [75]. 48
JAX-WS	Java API for XML - Web Services [104]. 48
JPA	Java Persistence API [54]. 43
JPQL	Java Persistence Query Language [54]. 49
KM3	Kernel Meta Meta Model [99, 165]. 51
LSF	loi de sécurité financière [126]. 69
MDA	model-driven architecture [130]. 3

MDD	model-driven development. ii, 3–6, 13, 17–19, 24–31, 35–40, 52–54, 56–59, 63, 67, 73, 81–83, 85, 86, 94–96, 99, 100
MDE	model-driven engineering. 2, 3, 5, 8–10, 35, 38, 39, 47, 54, 66, 67, 69, 83
MDR	NetBeans metadata repository [122]. 50, 51
MEP	message exchange pattern [39]. 62
MiFID	Markets in Financial Instruments Directive [60]. 69
MOF	Meta-Object Facility [131]. 3, 15, 39, 51, 52
MORSE	Model-Aware Service Environment [82, 87, 86]. 8, 10, 25, 27, 29, 33, 35–40, 42–44, 46–54, 56–61, 63, 64, 66, 67, 70, 85, 89–92, 94–96, 98, 99, 101–104
MVNO	mobile virtual network operator. 14, 27, 29
oAW	openArchitectureWare [138]. 40
OCL	Object Constraint Language [133]. 14, 28
OMG	Open Management Group. 2, 3, 13, 39, 42
OMT	object-modeling technique. 2
OWL	Web Ontology Language [17]. 51
PDF	Portable Document Format [93]. 76
PIM	platform-independent model. 3, 4, 73
PSM	platform-specific model. 3, 4, 73
QoS	quality of service. 70, 98, 99
RBAC	role-based access control. 4
RDBMS	relational database management system. 43, 51
RE	requirements engeneering. 99
REST	representational state transfer. 48, 101
RSS	Really Simple Syndication [179]. 66, 67
SCM	software configuration management [88, 92]. 43, 52

SE	software engineering. 2, 3, 38, 103
SLA	service level agreement. 7, 98, 99
SOA	service-oriented architecture. 5–8, 10, 35, 36, 46, 48, 56, 59, 63, 66, 69–71, 75, 77–81, 88, 89, 92, 96, 98, 100–103
SOC	service-oriented computing. 5, 101, 103
SoC	separation of concerns. 3, 69
SoD	separation of duties. 70
SOX	Sarbanes-Oxley Act [49]. 70, 85, 90, 93, 104
SRS	software requirement specification. 81
SVN	Subversion [159]. 41
U.S.	United States. 93
UC	unit of comparision (cf. [137, p.3]). 52
UI	user interface. 94
ULS	ultra-large-scale. 66
UML	Unified Modeling Language [134]. 2, 3, 15, 30, 31, 39, 51, 52
URI	Uniform Resource Identifier [77]. 50–52, 58
URL	Uniform Resource Locator [77, 22]. 50–52
URN	Uniform Resource Name [77, 120]. 44
UUID	Universally Unique Identifier [113, 98]. 9, 36, 37, 39, 40, 43, 44, 46, 49, 50, 52, 53, 57–61, 63, 86, 87, 89, 92, 94, 95, 102, 126
UV	unit of versioning (cf. [137, p.4]). 43, 50–52
VbMF	View-based Modeling Framework [173, 84, 171, 172]. 27, 70, 75, 77, 79, 81, 86, 104
VCS	version control system. 31, 39, 41, 52
VDE	validation/deployment/execution. 61, 62, 104
VIID	version-independent UUID. 44, 49
VSID	version-specific UUID. 44, 49
WADL	Web Application Description Language [74]. 58
WSDL	Web Services Description Language [40]. 35, 58, 62

XMI	XML Metadata Interchange [135, 95]. 39, 49, 52
XML	Extensible Markup Language [29]. 48, 49, 59–61, 89, 101
XPath	XML Path Query Language [21]. 59

Index

Ta‘id Holmes studied Computer Science at the Vienna University of Technology and Organic Chemistry at the Vienna University of Technology, CPE Lyon and Université Claude Bernard Lyon 1. He received a Dipl.-Ing. from the Vienna University of Technology in Software Engineering/Internet Computing and a DEA (Diplôme d'Études Approfondies) in Chimie Organique Fine from the Université Claude Bernard Lyon 1. Since March 2007 he is a guest lecturer at the Vienna University of Economics and Business Administration. In May 2007 he joined the Distributed Systems Group of the Institute of Information Systems at the Vienna University of Technology as a research assistant.