Monitoring and Analyzing Service-Based Internet Systems through a Model-Aware Service Environment

Ta'id Holmes[1], Uwe Zdun[1], Florian Daniel[2], and Schahram Dustdar[1]

[1] Distributed Systems Group, Institute of Information Systems
Vienna University of Technology, Vienna, Austria
{tholmes, zdun, dustdar}@infosys.tuwien.ac.at

[2] Information Engineering and Computer Science Department
University of Trento, Trento, Italy
daniel@disi.unitn.it

Abstract 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. By making the information in these models accessible at runtime, we provide better means for analyzing and monitoring the service-based systems. We propose a model-aware repository and service environment (MORSE) to support model access and evolution at both design time and runtime. MORSE 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.

1 Introduction

In the Internet of services, systems get increasingly complex. Often it is hard to manage, analyze, and monitor such complex service-based systems at runtime. This is, (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. [1,2]) and requirements (cf. [3,4]) today are modeled with precisely specified and detailed models. One of these reasons is the increasing use of model-driven development (MDD) [5] 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. We propose 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.

As an illustrative case for system requirements, consider compliance to regulations: A business information system needs to comply with regulations, such as Basel II [6] or the Sarbanes-Oxley Act (SOX) [7]. 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 paper, we propose to address these issues using an architecture and software environment called the Model-Aware Repository and Service Environment (MORSE) [8]. 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 time 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.

This paper is structured as follows: In the next section we will give a motivating example and illustrate the monitoring architecture employed for our approach. In Section 3 we present MORSE the Model-Aware Repository and Service Environment and describe how Internet-based systems can be enhanced with traceability information for monitoring and analyzing these systems. Next, in Section 4 we illustrate our work with a case study. Performance and Scalability evaluations are then presented in Section 5. Section 6 compares our approach to related work, and in Section 7 we conclude and refer to future work.

2 A Model-Aware Service Environment in Compliance Monitoring and Analysis

First we outline a concrete example motivating the need for a model-aware repository and service environment for monitoring and analysis purposes. We consider the problem of monitoring and analyzing business processes to identify compliance violations. In particular, we focus on a monitoring infrastructure that observes a business IT system while it is running business processes modeled, for instance, as BPEL [9] processes. Using a dedicated model for compliance concerns, the system knows about how to identify violations. A simple example of a compliance rule is the four eyes principle, which requires that a single operation in a business process must be performed by two different actors. If the monitoring infrastructure observes a violation of a compliance concern, it reports it to the user via a compliance dashboard.

Upon the detection of a violation, the user typically wants to understand which process has caused the violation and why. Hence, the user should access the process model that has caused the violation. However, sometimes the process model is not the only relevant information or not the root cause of the violation. Other models linked to the process model might carry the answer to the violation; hence, they must be accessed, too. Examples are the model specifying the compliance rule that has been violated or the models of the processes that have invoked the process leading to the violation.

Finally, once the root cause has been identified, it is likely that the user will fix the affected models. Then, the corrected models should from then on be used for new instances of the business processes. Here, a transparent versioning support is crucial, as it is typically not possible to automatically migrate running process instances from one model to another. Old model versions must be supported as long as instances of them are running, unless the execution of such instances is halted.

To enable using models at runtime as described in this scenario, we propose the following runtime features:

Model-aware repository and service environment The repository contains all models of the system. These can be queried and changed via service-based interfaces at runtime (see also Figure 2).

Eventing infrastructure In our approach, the code setting up the emission of events is generated via MDD from process and compliance models. That is, the process engine is instrumented to emit events, such as *process started*, *process ended*, or *process activity entered*. Each of these events includes a unique identifier to the model artifact, such as *process model P* or *activity A in the process model P*, that is the source of the event.

Monitoring infrastructure A monitoring infrastructure is required to continuously observe and analyze emitted events to check compliance. Runtime requirements-monitoring systems, such as proposed by Feather et al. [10] and Skene and Emmerich [3], can be integrated with MORSE and used for the compliance checking. If a violation occurs, the model responsible for the violation can easily be identified using the identifier contained in the event(s) that caused the violation.

In the MORSE repository, the various models are linked via relationships. Hence, the root cause of the violation can be determined by traversing these relationships. For instance, a process model can be annotated in different models exogenously with compliance concerns, and the monitoring infrastructure needs to resolve what concern has been violated. To do so, another service-based request is sent to the repository for retrieving the compliance concerns that annotate the source model. In a number of interactive steps, the root cause of a violation can be detected.

Figure 1 shows the application of our approach for our illustrative case study in an event-driven architecture [11] for compliance monitoring combining both online monitoring (on-the-fly analysis of events at runtime) and offline monitoring (analysis of audit trails and event logs) in an service-oriented environment. Online monitoring and on-the-fly event analysis is necessary to react as quickly as possible to violations. However, not all violations of compliance in service-based Internet systems can be determined online (consider, e.g., long-running processes). Such violations typically require offline analysis such as an investigation of the event history.

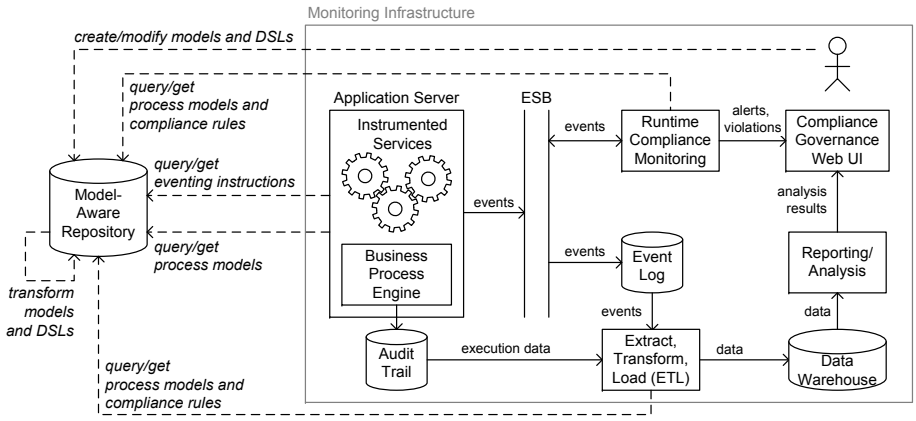

Figure 1. MORSE Combined with an Online and Offline Monitoring Infrastructure

Figure 1 shows a representative example of a monitoring infrastructure combining the two monitoring approaches with MORSE: The execution of services and business processes emits events, which provide information about the concrete execution of services and process instances. Events are, therefore, emitted by the business process engine and by the instrumented services and published via a central enterprise service bus (ESB). The ESB provides publish/subscribe support for events consumed by the monitoring components. In the proposed architecture, the event log component is subscribed to any event that is relevant for monitoring and further processing. Additional process execution data (e.g., data that is not carried with events, such as engine-internal events) are stored in a dedicated audit trail.

Starting from the event log and the audit trail, the analysis components such as the data warehouse (that stores all execution data), the extract/transform/load (ETL) procedures (that are responsible for extracting the data needed for the compliance analyses from the event log and the audit trail, transform them into the data format of the data warehouse, and load them into the warehouse), and the compliance governance Web user interface (UI), which includes a compliance governance dashboard for human users.

Processes and compliance concern models are stored in the MORSE repository and deployed onto the compliance monitoring infrastructure for execution, monitoring, and analysis. Specifically, the process engine and instrumented services query the model-aware repository for process models and eventing instructions. Eventing instructions are automatically generated from the compliance concerns in the repository. The runtime compliance monitoring component and the ETL procedures query the repository for process models and compliance concern models for compliance evaluation. Finally, the user of the compliance governance UI creates and modifies models and compliance concerns on-the-fly, directly working on the MORSE repository.

3 Model-Aware Repository and Service Environment

The integration of MORSE into the compliance governance infrastructure with its online and offline monitoring features poses a variety of requirements to its model management capabilities. As most stringent we specifically highlight the following:

- It is necessary to *store*, *deploy*, *maintain* and *evolve* process and service models as well as compliance annotations (expressed via dedicated DSLs) in a way that allows one to keep consistent *relationships* among these artifacts, which are typically subject to change over time. For instance, it is necessary to be able to associate a set of compliance rules to a process model and to correctly maintain such association even after the modification of the process model or the addition, modification or deletion of individual compliance rules.
- For the online analysis of compliance violations, it is necessary to be able to *drill down* from high-level process models down to individual activities or event annotations, associated with the process model. It is therefore necessary that the repository is aware of the hierarchical composition of process definitions and that each element in the model can be properly indexed. Given a violation of a compliance rule in a process instance, it might, for example, be necessary to retrieve the process definition and to drill down to the specific activity that caused the violation, in order to understand how to mitigate future violations.
- Finally, given its use at runtime of the monitoring infrastructure, *fast response times* and *high query throughput* are paramount. Individual compliance rules associated with a given process model might, for instance, be queried at each instantiation of the process, in order to set up the runtime compliance monitoring module (based on complex event processing [11]).

MORSE supports the above compliance governance scenario (and similar cases) in that it suitably addresses these requirements, going beyond simple file system based model management. In particular, the main features of MORSE that simplify the development of the monitoring infrastructure include:

- 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 (for details see also Figure 5 in [8]).
- *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 details see also Table I in [8]). For reflectively exploiting the relationships of MDD artifacts, the MORSE information retrieval 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, complex queries can be constructed in an interactive, stepwise manner.

– The MORSE repository provides *versioning* capabilities not only to the artifacts, but also to their relationships. 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 all their model instances are either deleted or migrated to the new model version.

Figure 2 shows the internal architecture of MORSE that can be used by various clients via its Web service interfaces. 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.

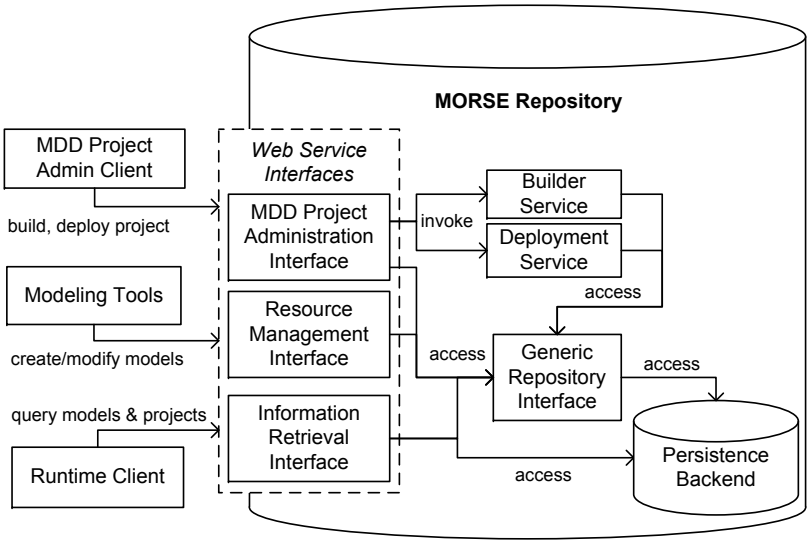

Figure 2. MORSE Architecture

The model repository is the main component of MORSE and has been designed with the goal to abstract from specific technologies. Thus, while concepts are taken from, e.g., UML [12], and also version control systems such as Subversion [13], MORSE is particularly agnostic to specific modeling frameworks or technologies. The model repository itself has been realized with technologies such as Eclipse Modeling Framework (EMF) [14], Teneo [15], EclipseLink [16], PostgreSQL [17], and Apache CXF [18].

MORSE can also generate Web services for domain concepts that are expressed in EMF models. This is, for every concept a service is generated with basic *create*, *retrieve*, *update*, *delete* and *query* operations. For the different relations between con-

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

Listing 1.1. Populating MORSE with Instances of Compliance Domain Concepts

cepts appropriate *add* and *remove* operations are generated in addition for associating and deassociating role instances.

Besides these service implementations, MORSE provides developers with Java libraries that are distributed using a Maven [19] repository. Listing 1.1 shows an example from a script that populates MORSE with instances of compliance domain concepts using the generated libraries. In the example a compliance source that relates to SOX Section 409 as well as a compliance risk are associated with a compliance requirement.

3.1 The MORSE Approach for Service-Based Internet Systems

Our approach applies MDD to generate services and process code, deployment artifacts, and monitoring directives. Usually, in MDD there are no backward or traceability links in the sense that the generated source code "knows" from which model it has been created. For correlating model instances or code at runtime with source models or model instances, respectively, traceability of model transformations is essential.

To achieve traceability, models (as the output of the generator) can hold a reference to their source models. As MDD artifacts in MORSE repositories are identifiable by Universally Unique Identifiers (UUIDs) [20], MORSE annotates the destination models with the UUIDs of the models. The generator can automatically weave references to these UUIDs into the generated source code or configuration instructions, so that the corresponding models can be identified and accessed from the running system. Please note, that UUIDs are transparently created and used in the MORSE repository (e.g., the *create* operation returns the assigned UUID). Particularly, a user or developer (cf. Listing 1.1) does not necessarily have to specify UUIDs for interacting with MORSE.

The code in Listing 1.2 shows a generated BPEL process that contains traceability information 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.

```
<process name="ReportIntrusion">
  <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]"
        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      <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 1.2. BPEL Process with an Extension for MORSE Traceability

4 Case Study: Compliance to Regulations

Let us consider an industrial case study in which MORSE has been applied: A US credit card company that wants to comply with Section 409 (Real Time Issuer Disclosures) of SOX [7]. 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 3. 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 3 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).

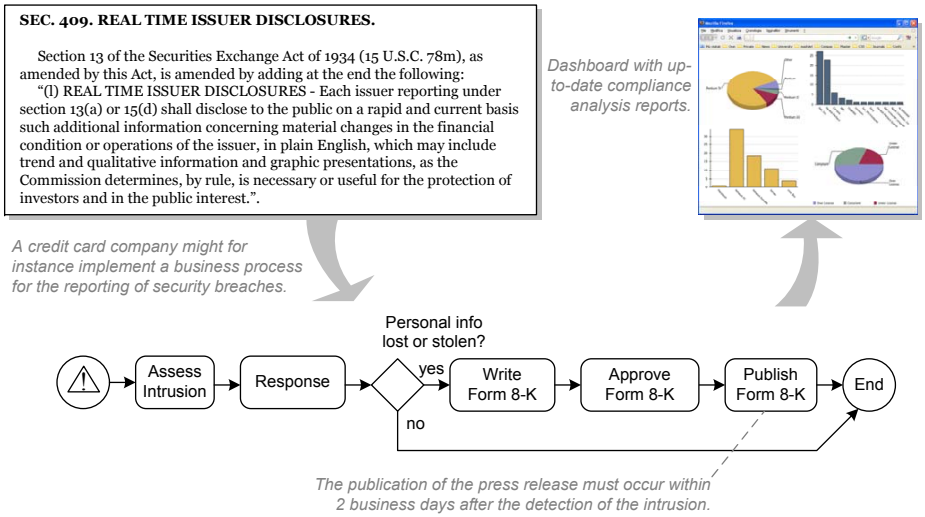

Figure 3. SOX Example

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 analyses to understand the reason for specific compliance violations is needed.

We assume that the monitoring infrastructure in Figure 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 analyses 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.

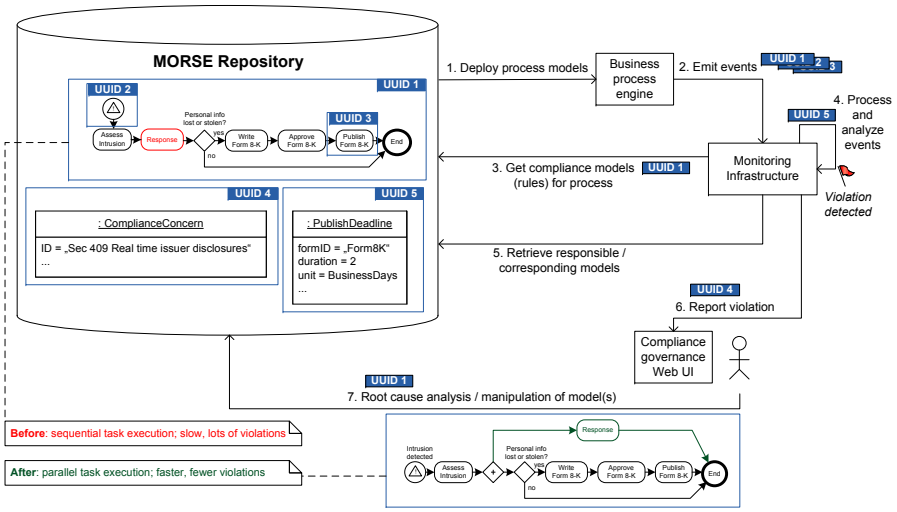

Figure 4. Resolved SOX Example

Figure 4 illustrates the interplay of MORSE, the monitoring infrastructure, and the compliance governance Web 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 `ComplianceConcern` 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 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 concern 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 concern 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.

5 Performance and Scalability Evaluation

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 5. 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 5, 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 lookups, e.g., huge MORSE repositories with more than $\approx 2^{17}$ models cannot perform 10^6 lookups 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 Related Work

In our approach we assume that a monitoring infrastructure is used throughout the architecture for observing and analyzing runtime events (cf. Section 2). While such monitoring infrastructure can be integrated with MORSE and used for the compliance checking, our work particularly focuses on relating to models, the monitored systems have been generated from. 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. In this section we refer and relate to work in the areas of runtime requirements-monitoring and model management in form of model repositories.

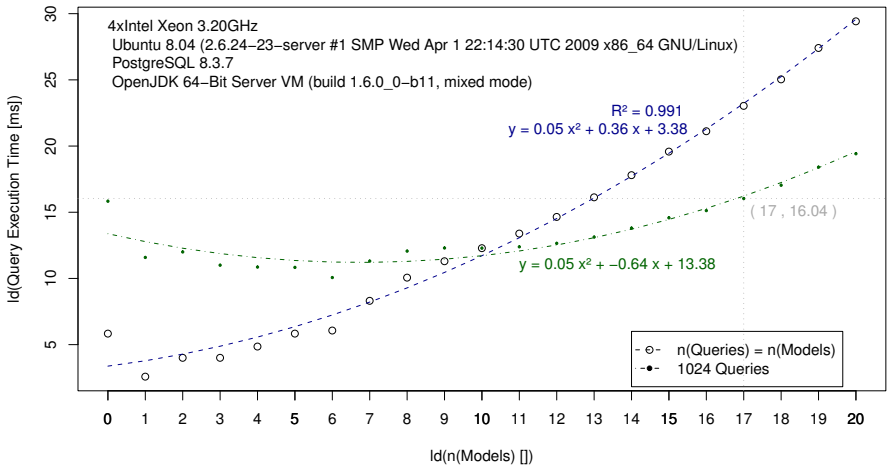

Figure 5. MORSE Performance and Scalability

Feather et al. [10] discuss an architecture and a development process for monitoring system requirements at runtime. It builds on work on goal-driven requirements engineering [21] and runtime requirements monitoring [22].

Skene and Emmerich [3] 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 meta-data 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 3), 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. [4] present a MDD framework and methodology for creating Business Performance 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 quality of service concerns as found in service level agreements 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.

Another model-based design for the runtime monitoring of quality of service aspects is presented by Ahluwalia et al. [23]. Particularly, an interaction domain model and an infrastructure for the monitoring of deadlines are illustrated. In this 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 only related to these in a sense that it reflects them. This is, it is not a source model for the model-driven development of the services. In contrast, MORSE manages and is aware of the real models, systems are generated from. This allows for root cause analysis and evolution of as demonstrated in the presented case study (see Figure 4).

Besides the monitoring of runtime requirements in form of compliance concern models, 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). It abstracts from modeling technologies and its UUID-based implementation allows for a straightforward identification of models and model elements. Other model repositories such as ModelBus [24] and ModelCVS [25] primarily aim at model-based tool integration. AMOR [26,27] and Odyssey-VCS 2 [28] particularly have a focus on the versioning aspect of model management (see also [29]), e.g., for the conflict resolution in collaborative development (cf. [30]). These works mainly focus on the design time: e.g., ModelBus addresses the heterogeneity and distribution of modeling tools and focuses on integrating functionality such as model verification, transformation, or testing into a service bus. MORSE, in contrast, 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.

7 Conclusion

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 model-driven development 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. As a first step into this direction, in this article we have presented MORSE, an implementation of a model-aware repository and service environment concept that treats models as first-class citizens at runtime of a service-based system. The MORSE approach significantly eases the management of complex service-based systems by improving the analyzability, e.g., of monitoring data. This has been demonstrated in an industrial case study for compliance management. 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.

Acknowledgments

This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

References

1. Mayer, P., Schröder, A., Koch, N.: MDD4SOA: Model-driven service orchestration. In: EDOC, IEEE Computer Society (2008) 203–212
2. Zdun, U., Hentrich, C., Dustdar, S.: Modeling process-driven and service-oriented architectures using patterns and pattern primitives. TWEB **1**(3) (2007)
3. Skene, J., Emmerich, W.: Engineering runtime requirements-monitoring systems using mda technologies. In Nicola, R.D., Sangiorgi, D., eds.: TGC. Volume 3705 of Lecture Notes in Computer Science., Springer (2005) 319–333
4. Chowdhary, P., Bhaskaran, K., Caswell, N.S., Chang, H., Chao, T., Chen, S.K., Dikun, M.J., Lei, H., Jeng, J.J., Kapoor, S., Lang, C.A., Mihaila, G.A., Stanoi, I., Zeng, L.: Model driven development for business performance management. IBM Systems Journal **45**(3) (2006) 587–606
5. Völter, M., Stahl, T.: Model-Driven Software Development: Technology, Engineering, Management. Wiley (2006)
6. Bank for International Settlements: Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version. `http://www.bis.org/publ/bcbsca.htm` (June 2006) [accessed in February 2010].
7. Congress of the United States: Public Company Accounting Reform and Investor Protection Act (Sarbanes-Oxley Act), Pub.L. 107-204, 116 Stat. 745. `http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html` (July 2002) [accessed in February 2010].
8. Holmes, T., Zdun, U., Dustdar, S.: MORSE: A Model-Aware Service Environment. In Kirchberg, M., Hung, P.C.K., Carminati, B., Chi, C.H., Kanagasabai, R., Valle, E.D., Lan, K.C., Chen, L.J., eds.: Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC), IEEE (December 2009) 470–477
9. Organization for the Advancement of Structured Information Standards: Web service business process execution language version 2.0. OASIS Standard, OASIS Web Services Business Process Execution Language (WSBPEL) TC (January 2007)
10. Feather, M., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behavior. In: Software Specification and Design, 1998. Proceedings. Ninth International Workshop on. (Apr 1998) 50–59
11. Michelson, B.: Event-Driven Architecture Overview: Event-Driven SOA Is Just Part of the EDA Story. `http://www.omg.org/soa/Uploaded%20Docs/EDA/bda2-2-06cc.pdf` (February 2006) [accessed in February 2010].
12. International Organization for Standardization: ISO/IEC 19501:2005 information technology – open distributed processing – unified modeling language (UML), v1.4.2. `http://www.omg.org/cgi-bin/doc?formal/05-04-01` (April 2005) [accessed in February 2010].
13. The Apache Software Foundation: Apache Subversion. `http://subversion.apache.org` (2000) [accessed in February 2010].
14. The Eclipse Foundation: Eclipse Modeling Framework Project (EMF). `http://www.eclipse.org/modeling/emf/` (2002) [accessed in February 2010].

15. The Elver Project: Teneo. http://www.eclipse.org/modeling/emf/?project=teneo (2005) [accessed in February 2010].
16. The Eclipse Foundation: Eclipse Persistence Services Project (EclipseLink). http://www.eclipse.org/eclipselink (2008) [accessed in February 2010].
17. PostgreSQL Global Development Group: PostgreSQL. http://www.postgresql.org (1997) [accessed in February 2010].
18. The Apache Software Foundation: Apache CXF: An Open Source Service Framework. http://cxf.apache.org [accessed in February 2010].
19. The Apache Software Foundation: Apache Maven. http://maven.apache.org [accessed in February 2010].
20. 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. http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf (September 2004) [accessed in February 2010].
21. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. **20**(1-2) (1993) 3–50
22. Cohen, D., Feather, M.S., Narayanaswamy, K., Fickas, S.S.: Automatic monitoring of software requirements. In: ICSE '97: Proceedings of the 19th international conference on Software engineering, New York, NY, USA, ACM (1997) 602–603
23. Ahluwalia, J., Krüger, I.H., Phillips, W., Meisinger, M.: Model-based run-time monitoring of end-to-end deadlines. In Wolf, W., ed.: EMSOFT, ACM (2005) 100–109
24. Sriplakich, P., Blanc, X., Gervais, M.P.: Supporting transparent model update in distributed case tool integration. In Haddad, H., ed.: SAC, ACM (2006) 1759–1766
25. Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., Schwinger, W.: Towards a semantic infrastructure supporting model-based tool integration. In: GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management, New York, NY, USA, ACM (2006) 43–46
26. Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., Wimmer, M.: AMOR – towards adaptable model versioning. In: 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS '08. (2008)
27. Brosch, P., Langer, P., Seidl, M., Wimmer, M.: 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) 55–60
28. Murta, L., Corrêa, C., Prudêncio, J.G., Werner, C.: 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, New York, NY, USA, ACM (2008) 25–30
29. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. IJWIS **5**(3) (2009) 271–304
30. Brosch, P., Seidl, M., Wieland, K., Wimmer, M., Langer, P.: 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) 207–214