Automating the Management and Versioning of Service Models at Runtime to Support Service Monitoring

Ta'id Holmes*, Uwe Zdun†, and Schahram Dustdar*
* *Distributed Systems Group, Institute of Information Systems*
Vienna University of Technology, Austria
{tholmes, dustdar}@infosys.tuwien.ac.at
† *Software Architecture Group, Faculty of Computer Science*
University of Vienna, Austria
uwe.zdun@univie.ac.at

***Abstract*—In a model-driven service-oriented architecture (SOA), the services are in large parts generated from models. To facilitate monitoring, governance, and self-adaptation 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 metamodel 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. To address this problem, this paper introduces model-aware services that work with models at runtime. These services are supported using a service environment, called MORSE. Hiding the complexity of implicit versioning of models from users while respecting the principle of Universally Unique Identifiers (UUIDs), it realizes a novel transparent UUID-based model versioning technique. It uses the model-driven approach to automatically generate and deploy MORSE services that are 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.**

***Keywords*-Service Management, Service Runtime, Service Versioning, MDE, Model Repository, Model Versioning, UUID, SOA**

Modern software systems are becoming increasingly complex. Some reasons for this trend are that they unify different technologies and services are exposed in heterogeneous environments. Model-driven engineering (MDE) 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. 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 interesting as models can be used for communication of the different stakeholders. Also in a couple of scenarios such as service monitoring or adaptation (cf. [1]) 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 on 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 call services that dynamically work with models *model-aware*.

There are a couple of obstacles prior to the adoption of models in service-oriented architectures (SOAs). First, in a distributed and heterogeneous environment the identification and retrieval of models from services is challenging. This is because of the need for unique identifiers across the SOA and the lack of a common, unified, and service-based access to models. Second, if a metamodel for service models changes, dependent model-aware services need to be adapted, recompiled, and redeployed to work with the new version. 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 becomes difficult with 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, coexistent versions.

Our approach utilizes a service environment, called Model-Aware Service Environment [2], [3] (MORSE), for model publication and lookup. To deal with model evolution, we introduce a novel transparent versioning approach, which allows different services to transparently use different versions of models or model elements at runtime. To make the approach usable, MORSE *automatically* generates the services needed to query and traverse models at runtime when a service model is stored in MORSE. Universally Unique Identifiers [4] (UUIDs) are used to enable monitoring and adaptation services to access the *correct* model or model element in the required *version*. For example, code for raising events containing the unique identifiers can be generated into model-aware services, enabling other components to monitor and adapt the services, and relate to the models in the particular versions during monitoring and adaptation tasks. 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 and that uses a model repository to support models at runtime in SOAs that can evolve transparently.

The remainder of this paper is structured as follows: In Section I a motivating scenario is described that concretizes the context and prerequisites. The approach for the management of model versions and for supporting services to dynamically work with models is described in Section II. Next, in Section III a solution for a transparent UUID-based versioning is presented. The automatically generated MORSE services are described in Section IV. Section V compares to related work, lessons learned are presented in Section VI, and Section VII concludes the paper.

I. MOTIVATION: MONITORING IN A SOA

Before we describe our approach, a motivating scenario is given that exemplifies a general setup and is the starting point for further discussion. Thus, we depict the context and introduce the prerequisites of our work. In this scenario, MDE is applied for developing 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 event notification. Hence, the services emit events and the monitor processes these (cf. [5] 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 further events than only 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 the next step, the monitor can use the information from the models the events relate to. For example, this can be the models from which the services have been generated. In a more elaborative setup, such events can also relate to other models as well. Next, to facilitate the governance, management, and adaptation of the SOA the monitor needs to consider the information from the models. Typically, this is done by reflecting on the value or state of model elements, certain semantics have been attributed to. For example, the monitor can compare values against thresholds. Ideally, the monitor is able to retrieve relevant models dynamically. 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 ideally are captured in the form of models, too (cf. [1] 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 but also operates on models.

If a metamodel is modified that a monitor is dependent on, the monitoring service also has to be changed; i.e., it usually has to be 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 significant effort and this in turn impedes the use of models at runtime.

II. APPROACH OVERVIEW

We aim at avoiding the aforementioned obstacles by adopting models at runtime. With our approach models are consistently used in different layers and parts of the system. In this section, we specify the different requirements and introduce our approach. It leverages the service-based use of models at runtime employing an automatic generation from metamodels. This is complemented with a version management of models.

A. Management of Model Versions

All models (i.e., metamodels and conforming models (cf. [6])) are stored in a model repository. 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.

B. Supporting Services to Dynamically Work with Models

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

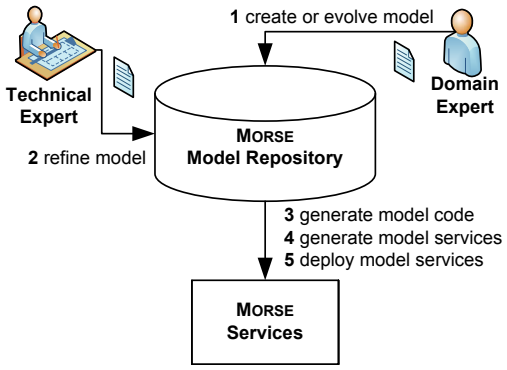

Figure 1. Generating MORSE Services for Models

For this, we automatically generate MORSE services for managing models as depicted in Figure 1. Usually, a domain expert starts to design a domain model (Step 1), i.e., a metamodel 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 stakeholders to formulate the models. Finally, in order to support the use of the resulting model in a SOA, MORSE services are automatically generated and deployed (Steps 3-5) that can be used by other services for sharing, storing, and retrieving models.

For facilitating development of a monitor, MORSE can also automatically generate ready to use service clients that interact with the MORSE services. This is demonstrated in Figure 2 (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 depicted in the figure (Step 3), they may be also automatically generated from models using MDE.

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

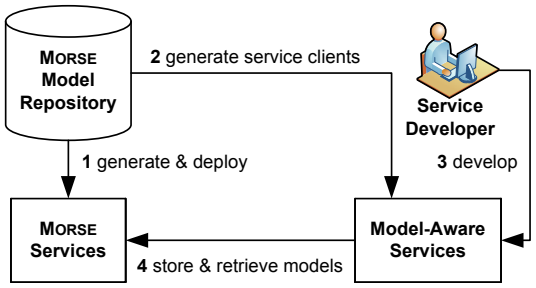

Figure 2. Supporting Model-Aware Services with Service Clients

III. TRANSPARENT UUID-BASED MODEL VERSIONING

A model contains model elements such as *classes*, *attributes*, and *references* (cf. Ecore [7] and EMOF [8] *classes* and *properties*). In the MORSE repository each instance of a ($M2$, metamodel) class corresponds to a table in a relational database [1]. An instance thereof (a $M1$ model) 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. [10, 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 for references the UUIDs of the corresponding model class instances are used.

For supporting the management of model versions during model evolution we propose a transparent versioning mechanism as realized in the MORSE repository. In addition to version-specific models, the latest version is always present in the form of a version-independent model (head) in MORSE. All of these models and its model-elements are identifiable by UUIDs. Hence, for their identification we distinguish between version-independent and -specific UUIDs. Thus, a versions-independent (resp. specific) UUID identifies models or model elements of the head (resp. a certain revision).

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.

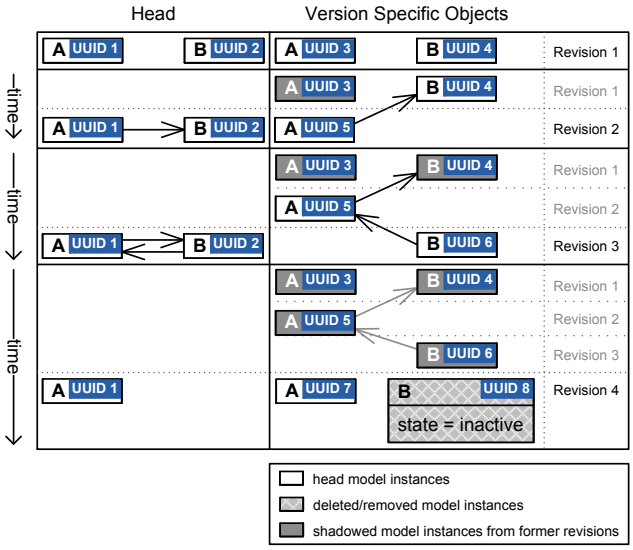

Figure 3. Transparent Versioning

Figure 3 depicts an example of 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. 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 updated. Note that a new version-specific object is created that references `B`, *overshadowing* 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 overshadowed. 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-

[1]The MORSE repository is realized with Teneo [9] that utilizes the Eclipse Modeling Framework [7] (EMF).

specific objects of that revision the reference from `A` still points to the now overshadowed `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 overshadows 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 1 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 1: Reconstructing a Revision of a Model from Version-Specific Objects

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

IV. MORSE Services

In this section we describe the MORSE services, that provide retrieve and storage functionalities for models in a SOA and realize the transparent model versioning as explained in the previous section.

These services are automatically generated from metamodels and exposed as XML and RESTful Web services. Model-aware services can thus choose between these implementations. Note that in addition to a service implementation MORSE also generates service clients (see also Step 2 of Figure 2) for using the MORSE services.

All of the generated software modules are organized as Maven [11] projects and distributed using a Maven repository. In this way, service developers such as displayed in Figure 2 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 that automates the retrieval and use of required modules.

With the model-driven generation, the deployment of services, and the distribution of service-clients 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 implementation of the Essential Meta Object Facility [8] (EMOF) $M3$ meta-metamodel – 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 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 model technologies and serialization formats. Support for further model 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 I
MORSE SERVICE OPERATIONS

Response	Operation	Description
boolean	exists	does a model with a UUID exist?
boolean	isHead	is the UUID version-independent?
UUID[]	list	all version-independent UUIDs
UUID[]	versions	version-specific UUIDs of a model
<*Class*>[]	query	search for models; support of various query parameters
<*Class*>	retrieve	a model is retrieved by UUID
UUID	create	UUID is version-independent
UUID	update	UUID is version-specific
UUID	delete	UUID is version-specific
UUID	list<*Role*>	UUIDs for a role
UUID	add<*Role*>	UUID is version-specific
UUID	remove<*Role*>	UUID is version-specific
UUID	clear<*Role*>	UUID is version-specific

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

that match specified query parameters. These parameters support `WHERE` and `ORDER` clauses (cf. [12]). For pagination also an index for the first result can be specified as well as the number of maximum results returned.

V. Related Work

The MORSE approach particularly focuses on the management of models of service-based systems and their accessibility during runtime for facilitating monitoring, governance, and self-adaptation. For this reason, a model repository with versioning capabilities is deployed (see Section III). It abstracts from modeling technologies and its UUID-based implementation allows for a straightforward identification of models and model elements. In this section we compare the MORSE repository against other model repositories and relate the MORSE approach to work in the field of monitoring.

A. Model Repositories

There are a number of related model repositories. Table II lists their methods of identification for models and model elements, indicates supported modeling technologies, the smallest unit of versioning (cf. [10, p.4]) (UV), and compares navigation and search capabilities. In the following we compare to the related work and to the data from the table.

The Adaptable Model Versioning [13], [14] (AMOR) model repository has a focus on the versioning aspect of model management (see also [23]), e.g., for the conflict resolution in collaborative development (cf. [24]). For this reason the smallest UV can be set to model elements. Models in AMOR are identifiable by Uniform Resource Locators [25], [26]. In addition, identifiers (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 [15] 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 [27], [28] (KM3), Web Ontology Language [29] (OWL), or Unified Modeling Language [30] (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.

The Connected Data Objects model repository [16] (CDO) is a server-client framework for EMF models. In EMF a model element is identifiable within a model [2] via a so called Uniform Resource Identifier [25] (URI)-fragment. Usually, although pluggable in CDO, a relational database management system (RDBMS) is used as a persistence backend (e.g., with Teneo [9]) 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 [31] communication framework and also aims to support the execution of server-side queries.

EMFStore [17] 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 [18] (MDR) was a Meta-Object Facility [8] (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 Odyssey-SCM (see below).

ModelBus [19], [20] 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 version control system (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.

Odyssey-SCM [21], [10] and Odyssey-VCS 2 [22] identify models using XML Metadata Interchange [32] (XMI) IDs. Model elements are identified with additional URI-fragments. While Odyssey-SCM used MOF 1.4, 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. [10, p.4]) (UV) and unit of comparision (cf. [10, p.3]) (UC) and make these customizable for the software configuration management [33] (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 [34] through exogenous "records of transformation".

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, the MORSE repository is the only

[2] A *Resource* (identifiable by a URL) in terms of EMF.

Table II
COMPARISON OF MODEL REPOSITORIES

Repository	Model Identification	Model Element Identification	Modeling Technology	Unit of Versioning	Model Navigation	Complex Search
AMOR [13], [14]	URL	ID	EMF	ANY	NO	NO
AtlanticZoo [15]	URL	NO	ANY	model	NO	NO
CDO [16]	URL	URI-Fragment	EMF	$M2$ class instance	YES	YES
EMFStore [17]	ID	ID	EMF	ANY	YES	NO
MDR [18]	ID	URI-Fragment	MOF 1.4	ANY	NO	NO
ModelBus [19], [20]	URL	NO	ANY	model	NO	NO
MORSE [2], [3]	UUID	UUID	ANY	$M2$ class instance	YES	YES
Odyssey-SCM [21], [10]	ID	URI-Fragment	MOF 1.4	ANY	NO	NO
Odyssey-VCS 2 [22]	ID	URI-Fragment	EMF	ANY	NO	NO

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 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 MDE projects. That is, the MORSE repository comes with explicit support for the management of MDE projects and supports the MDE process, something that is not supported by many other repositories (e.g., workflows, that cover processes of MDE, have to be defined on top of ModelBus).

Finally, the MORSE services support model navigation as well as complex (server-side) queries which are important features for the dynamic use and lookup of models.

B. Monitoring

The topic of service (cf. [35], [5]) and process (cf. [36]) monitoring is well covered by the literature; e.g., in the areas of quality of service (QoS) and SLAs. Yet by applying a model-aware service environment 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. [35]. 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, 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, systems are generated from and/or relate to. This allows for supporting the monitoring and adaptation in SOAs as outlined.

Commuzi et al. [5] explicate the link between SLA negotiation and monitoring of complex service-based systems. While being specific to SLAs, 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 clients.

In contrast to these works, the 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, SLAs, or compliance. 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 benefits in the areas of evolution and management.

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 MDE 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. Apart from the model repository, MORSE with its model-aware services establishes a service-oriented approach that has not yet been presented at large.

None of the mentioned model repositories offers an integration scheme for runtime events or the automated model generation and deployment capabilities. As shown in this paper, 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 or at least UUID-based identification and provides a model-driven component for runtime generation and deployment.

VI. DISCUSSION

In this paper, we have presented – to the best of our knowledge – the first model repository based approach to

support models at runtime in SOA. In our experiences, especially in large scale systems, such as SOA, the maintainability of models at runtime is difficult, as the system continuously evolves as it runs. 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 versioning approach to practically deal with this situation. Even for frequently changing central metamodels, on which many services, some developed by third parties, are based, our approach is easily applicable. 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 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, we needed to extend the MORSE repository to automatically generate and deploy the MORSE services in the background. In this way, every time a new metamodel version is created, automatically MORSE services are generated and deployed that enable users to easily deal with these versions. No other model management approach supports the automatic generation of specialized traversal and query services upon deployment of the metamodel 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 SOA, one final problem needed to be solved: The unambiguous identification of models, model elements, and versions thereof. We use UUIDs for this task 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.

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 deal with only the frontend parts of the MORSE environment for using models at runtime. In the model-aware services there is only a small performance overhead for raising the events, which is often needed anyway. For instance, in the context of compliance the events must be raised and logged for legal auditing purposes.

VII. CONCLUSION

In this paper we identified and proposed solutions to basic requirements for using models for service monitoring and adaptation. First of all, we addressed the need for the management of different model versions with a transparent UUID-based model versioning in MORSE. For facilitating services to work with models at runtime, MORSE services are automatically generated and deployed. 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 also generates and distributes appropriate service clients.

ACKNOWLEDGMENTS

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

REFERENCES

[1] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *MoDELS*, ser. Lecture Notes in Computer Science, A. Schürr and B. Selic, Eds., vol. 5795. Springer, 2009, pp. 468–483.

[2] T. Holmes, U. Zdun, and S. Dustdar, "MORSE: A Model-Aware Service Environment," in *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC)*, M. Kirchberg, P. C. K. Hung, B. Carminati, C.-H. Chi, R. Kanagasabai, E. D. Valle, K.-C. Lan, and L.-J. Chen, Eds. IEEE, Dec. 2009, pp. 470–477.

[3] T. Holmes, "Model-Aware Service Environment (MORSE)," Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Sep. 2008, [accessed in June 2012]. [Online]. Available: http://www.infosys.tuwien.ac.at/prototype/morse

[4] 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," Sep. 2004, [accessed in June 2012]. [Online]. Available: http://itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf

[5] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, and R. Yahyapour, "Establishing and Monitoring SLAs in Complex Service Based Systems," in *ICWS*. IEEE, 2009, pp. 783–790.

[6] J. Bézivin, "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[7] The Eclipse Foundation, "Eclipse Modeling Framework Project (EMF)," The Eclipse Foundation, 2002, [accessed in June 2012]. [Online]. Available: http://eclipse.org/modeling/emf

[8] Object Management Group, Inc., "Meta-Object Facility," Apr. 2002, [accessed in June 2012]. [Online]. Available: http://omg.org/mof

[9] The Eclipse Foundation, "Teneo," The Eclipse Foundation, 2005, [accessed in June 2012]. [Online]. Available: http://wiki.eclipse.org/Teneo

[10] H. L. R. Oliveira, L. G. P. Murta, and C. M. L. Werner, "Odyssey-VCS: a flexible version control system for UML model elements," in *SCM*. ACM, 2005, pp. 1–16.

[11] The Apache Software Foundation, "Apache Maven," The Apache Software Foundation, [accessed in June 2012]. [Online]. Available: http://maven.apache.org

[12] L. DeMichiel, "Java™ Persistence 2.0," Java Community Process, Java Specification Request 317, Dec. 2009, [accessed in June 2012]. [Online]. Available: http://jcp.org/en/jsr/detail?id=317

[13] K. Altmanninger, G. Kappel, A. Kusel, W. Retschitzegger, W. Schwinger, M. Seidl, and M. Wimmer, "AMOR – towards adaptable model versioning," in *1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS '08*, 2008.

[14] P. Brosch, P. Langer, M. Seidl, and M. 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.

[15] AtlanMod, "Atlantic Zoo," [accessed in June 2012]. [Online]. Available: http://www.emn.fr/z-info/atlanmod/index.php/Zoos

[16] The Eclipse Foundation, "Connected Data Objects (CDO) model repository," The Eclipse Foundation, 2005, [accessed in June 2012]. [Online]. Available: http://wiki.eclipse.org/CDO

[17] M. Kögel and J. Helming, "EMFStore: a model repository for EMF models," in *ICSE (2)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 307–308.

[18] M. Matula, "NetBeans metadata repository," NetBeans Community, [accessed in July 2009]. [Online]. Available: http://mdr.netbeans.org

[19] P. Sriplakich, X. Blanc, and M.-P. Gervais, "Supporting transparent model update in distributed case tool integration," in *SAC*, H. Haddad, Ed. ACM, 2006, pp. 1759–1766.

[20] A. Aldazabal, T. Baily, F. Nanclares, A. Sadovykh, C. Hein, and T. Ritter, "Automated Model Driven Development Processes," in *Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration*, 2008.

[21] L. G. P. Murta, H. L. R. Oliveira, C. R. Dantas, L. G. Lopes, and C. M. L. Werner, "Odyssey-SCM: An integrated software configuration management infrastructure for UML models," *Sci. Comput. Program.*, vol. 65, no. 3, pp. 249–274, 2007.

[22] L. Murta, C. Corrêa, J. G. Prudêncio, and C. 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*. New York, NY, USA: ACM, 2008, pp. 25–30.

[23] K. Altmanninger, M. Seidl, and M. Wimmer, "A survey on model versioning approaches," *IJWIS*, vol. 5, no. 3, pp. 271–304, 2009.

[24] P. Brosch, M. Seidl, K. Wieland, M. Wimmer, and P. 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.

[25] T. Hansen, T. Hardie, and L. Masinter, "Guidelines and registration procedures for new URI schemes," Feb. 2006, [accessed in June 2012]. [Online]. Available: http://ietf.org/rfc/rfc4395.txt

[26] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," Dec. 1994, [accessed in June 2012]. [Online]. Available: http://ietf.org/rfc/rfc1738.txt

[27] F. Jouault and J. Bézivin, "KM3: A DSL for Metamodel Specification," in *FMOODS*, ser. Lecture Notes in Computer Science, R. Gorrieri and H. Wehrheim, Eds., vol. 4037. Springer, 2006, pp. 171–185.

[28] The Eclipse Foundation, "KM3," The Eclipse Foundation, [accessed in June 2012]. [Online]. Available: http://wiki.eclipse.org/KM3

[29] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," Feb. 2004, [accessed in June 2012]. [Online]. Available: http://w3.org/TR/owl-ref

[30] Object Management Group, Inc., "Unified Modeling Language (UML)," Mar. 2000, [accessed in June 2012]. [Online]. Available: http://omg.org/spec/UML

[31] The Eclipse Foundation, "Net4j," The Eclipse Foundation, [accessed in June 2012]. [Online]. Available: http://wiki.eclipse.org/Net4j

[32] Object Management Group, Inc., "XML Metadata Interchange (XMI®)," [accessed in June 2012]. [Online]. Available: http://omg.org/spec/XMI

[33] International Organization for Standardization, "ISO 10007:2003 Quality management systems – Guidelines for configuration management," Mar. 2003, [accessed in June 2012]. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=36644

[34] C. Corrêa, L. G. P. Murta, and C. M. L. Werner, "Odyssey-MEC: Model Evolution Control in the Context of Model-Driven Architecture," in *SEKE*. Knowledge Systems Institute Graduate School, 2008, pp. 67–72.

[35] J. Ahluwalia, I. H. Krüger, W. Phillips, and M. Meisinger, "Model-based run-time monitoring of end-to-end deadlines," in *EMSOFT*, W. Wolf, Ed. ACM, 2005, pp. 100–109.

[36] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis, "Comprehensive Monitoring of BPEL Processes," *IEEE Internet Computing*, vol. 14, no. 3, pp. 50–57, 2010.