Living on the MoVE:
Towards an Architecture for a Living Models Infrastructure

Michael Breu, Ruth Breu, Sarah Löw
Quality Engineering Research Group
Institute of Computer Science, University of Innsbruck, Austria

Abstract—Developing, maintaining and operating large IT-systems is a complex and challenging task. Model-driven approaches can help to manage this complexity. However, during the life time of a system the models are not static but evolve through adaption to system changes or extensions. Therefore also models have to be organised and stored in a structured way. A model repository can manage the evolution of these models. However there are more complex requirements compared to classical source code or document repositories: Different modelling tools are involved, the consistency between models must be maintained, and finally, since various stakeholders are involved, changes must be propagated between models. In this paper we analyse these requirements and elaborate the basic architectural concepts for a Living Models infrastructure that supports the evolution of models.

Keywords—model versioning, model evolution, software engineering, change based process, modelling environment, tool integration;

I. INTRODUCTION

Model engineering is a widely accepted engineering discipline [1], and a lot of models are developed in manifold contexts in practice.

Models are within software projects a basis for (manual or automatised) development of software. In addition models may also be used in broader context, e.g. security models help to analyse and document security concepts in a critical IT-system, business process models document interleaved business processes and IT–landscape models help to manage IT–landscapes in the context of business processes and organisational structures.

Our own engineering experience in commercial projects shows that it is hard to keep models up–to–date, if no active continuous maintenance process is in force. Quite often models die slowly, i.e. they are getting gradually out–dated, since the physical systems are evolving, without further maintenance of the model.

The reasons for dying models are manifold. One major reason is that model management is in many cases centralised to a designated stakeholder. Thus only a limited set of persons can maintain it. In real life there are various stakeholders, that have the knowledge to maintain a model, but do not have the authorisation to update the model or no tool access. Also if the modelling tool supports multiple users, concurrent maintenance of a model is not always well supported. Neither do classical version management systems like subversion support the necessary conflict resolution for structured data like models. Finally, models can become quite complex, dependencies between modelled elements may be intricate, which makes it difficult to maintain the consistency of a model.

Facing this problem, ten principles were introduced in [2] to ensure an agile and flexible way to maintain models embedded in a change–driven engineering process. In this paper we will discuss the requirements for and impacts of these principles on the implementation of an Living Models infrastructure.

To this end we will conclude this section with a summary of the ten principles of Living Models. After this we will discuss related research approaches that cover aspects of the ten principles. This is followed by a short outline to a case study that we have conducted in order to further understand the requirements. In the main section we will introduce the major concepts of an infrastructure for Living Models, its basic architecture, and discuss the most important use cases for (meta–) model versioning and state management, before we conclude with an overview of the implementation of the initial infrastructure prototype.

Since the ten principles of Living Models[2] are the major motivation for this work, and to make the paper self–contained, we recapitulate these ten principles shortly in the following:

Persistence Models should be stored persistently and their evolution shall be supported.

Common System View All models of a current version should be related by a common system view. I.e. each model maintained in the repository is considered part of a common system model.

Information Consistency and Retrieval Based on the
common system meta-model consistency rules can be defined. Also new views on the information stored in the system model should be retrievable.

Bidirectional Information Flow between Models and Code and the Runtime System Code should be aware of the models and there should be an information flow from the code and the runtime system back to the models in order to enable monitoring and analysis.

Close Coupling of Models and Code The consistency between code and models is of major importance. Therefore changes to code artifacts should be reflected back to the model.

Model Element States Each model element can have a state that reflects certain aspects in its life cycle (e.g. a risk assessment may have the states draft, under review and final).

Change and Change Propagation The state of a model element can change between one system model version and the following. A state change may trigger other changes to the model, e.g. if an element in a final state depends on another element that has changed, its state may become also changed and be reset to draft.

Change-Driven Process The software development process in a Living Models environment is driven by change events, the states of the model elements and their interrelationships.

Stakeholder-Centric Modelling Environments The environment should support various stakeholders with different goals (and different requirements on the type and abstraction-level of the models, as e.g. an aggregated IT-landscape model, a risk model, or a database model).

Domains and Responsibilities A variety of stakeholders operates on the system model. In order to coordinate the work on the models in an organised way, there should be assigned responsibilities for each model element.

For details about the Ten Principles we would like to refer the reader to the original publication. In this paper we outline an infrastructure that implements those principles.

II. RELATED APPROACHES

The ten principles embrace research topics that are already part of intensive research such as model versioning and merging, and meta model management.

Model repositories with versioning support are a major topic of academic and industrial research projects as, e.g. ModelBus[3], or AMOR[4] show. Both projects try to establish a central repository where models and meta models can be stored and retrieved via adaptors from various tools. Bézivin et al.[5] coined the term MegaModel for the registry of models and meta models.

As soon as one starts versioning models in a distributed environment one has to face the problem of conflicts due to concurrent commits. Thus adequate model merging algorithms are an important topic. Kolovos et al. [6] has compared the most important algorithms, as e.g. EMF compare[7] or UMLDiff[8]. However there is still ongoing research in this area (see e.g. [9], [10]).

Related to the context of model repositories is the concept of model integration as a basis for (modelling) tool integration, as shown e.g. in Unicase[11], iRM[12], MOFLON[13], or (again) ModelBus[3]. AMMA[5] and AMOR[4]. Mainly the integration is achieved by the implementation of suitable adaptors that connect the tools with a model repository. The category of model integration also comprises the concept of model transformation as ATL[5] or as discussed by Strommer at al.[14] and round trip engineering which is already well established in many modelling tools, as e.g. eclipse MDT², MagicDraw³, or Rational Software Architect⁴.

Atkinson and Stoll [15] choose a different solution by managing a common system view together with derived views.

Managing a common system model also includes the aspect of meta model management as e.g. proposed by [16]. Since as meta models may evolve there is a need for tools that allow for the co-evolution of meta models together with the associated models as e.g. COPE[17], or the work concerned by EMF Refactor [18].

A topic that is not yet sufficiently covered by research is the topic of change-driven modelling. There are certain tools that include aspects of change-driven requirements engineering, as e.g. DOORS⁵ or in-Step change management⁶. These tools allow the requirements engineer to define state-based transition systems to model changes and their consequences to requirements.

The project MoVE (Model Versioning and Evolution) aims to establish an infrastructure to maintain Living Models. It does not concentrate on a single research topic, rather it tries to combines existing techniques (partially still under research, partially well established mechanisms) in a novel way to implement a change-driven process.

III. MOTIVATING EXAMPLE

To study the requirements of a Living Models infrastructure we conducted a case study based on the project health@net [19]. In the context of this project a security analysis of a virtually distributed electronic health record system was conducted.

The health@net project developed and maintained

- an enterprise model, defining the processes, information, services, hw nodes, etc. of the health@net system
- a security model, defining business security objectives, risks, security controls, etc. and associating them to elements in the enterprise model

²http://www.eclipse.org/modeling/mdt/?project=uml2
³http://www.magicdraw.com/
⁴http://www-01.ibm.com/software/awdtools/swarchitect/websphere/
⁵http://www-01.ibm.com/software/awdtools/doors/
⁶http://www.microtool.de/instep/en/Aenderungsmanagement%20mit%20in-
Step.asp
• a common meta model that defines the concepts in the security and enterprise model and their interrelations.

Both the meta model and the enterprise model was documented with MagicDraw. The security model was documented with a project specific application and persisted by a MySQL database. Based on the principle “Information Consistency and Retrieval”, the Living Models infrastructure helps to keep the consistency across different models and modelling tools alive.

The usage of different tools in this project not only originated from different requirements of each model–purpose, but also from the different involved stakeholders, such as the maintainer of the system meta model, the responsible for IT–systems (typically the organisation’s CIO or his/her deputy) and the responsible for security (e.g. the organisation’s CSO). To support their needs in the best way, the principle “Stakeholder–Centric Modelling Environments” is relevant.

The security analysis conducted in health@net is based on regular system checks as soon as an element of one of the above mentioned models changes or is initially added. These checks may change the (security) state, attached to each model element. To support this security analysis the principles “Model Element States”, “Change and Change Propagation” and “Change–Driven Process” have to be encompassed in the underlying infrastructure (for more details see Section IV-D).

IV. REQUIREMENTS FOR A LIVING MODELS INFRASTRUCTURE

In this chapter we will map the generic principles for Living Models to requirements of a working Living Models infrastructure.

A. Concepts

Before going into details we define some concepts, that we need for the definition of the requirements.

A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. This purpose determines what is to be included in the model and what is irrelevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail. (Taken from [20] Section 17.3.1). Models are expressed in specific concrete syntax, which can be graphical, textual or other suitable notation. Examples are state charts, business process diagrams, lists, trees or graph presentations, or any other domain specific language.

A model element is an atomic constituent of a model ([21]).

A system model is an abstraction of all relevant concepts and their relationships in a system. Thus a system model can be seen as a set of consistent models, or vice versa, a model represents a specific perspective on the system model. We speak of a partial model, if we want to emphasise that a model is part of the overall system model.

A meta model is a model that defines model element (types) and their relationships for expressing a model ([22]).

Expressing it in terms of the MOF layer model [23], the meta model refers to layer M2, the system model and the (partial) models refer to layer M1.

Being a little bit more formal we consider a MoVE meta model as a tuple $(MM,C,SM, m:SM \rightarrow Attributes(MM))$, where $MM$ is a meta model expressed as EMOF-model [21], together with a set $C$ of OCL statements, a set $SM$ of state machines (expressed as UML–based behavioural state machines [20], and a mapping $m$ that maps each state machine to a distinguished state attribute in the meta model. Set $C$ defines additional OCL constraints that cannot be expressed in a standard EMOF-model (as e.g. logical dependencies of model elements). $SM$ and $m$ form the definition of a change–driven model maintenance process, which will be explained in Section IV-D.

B. Basic Architecture

One of the major principles of a Living Models infrastructure is that various stakeholders can cooperate through a set of tools via the common system model. The common system model reflects the actual state of the physical system. Each stakeholder has its own tool (set) to maintain his/her own view of the complete picture. Figure 1 depicts this interaction.

Partial models from different stakeholders may overlap, e.g. in the security model of health@net risks are related to model elements in the enterprise model, i.e. two views
onto the system model may overlap. As long as this view is generated from the system model it can be easily handled as a projection on the relevant concepts of the system model, i.e. by just ignoring irrelevant concepts of the system model.

However, we accept that partial models are the basic artefacts to collect information about the real system, we have to merge modified partial models into the common system model. Depending on the type of underlying meta-meta-model as e.g. EMOF, UML, XML there exist different options for merging algorithms as explained in [24].

In order to reduce the set of potential conflicts, we define that each model element belongs to an “owner” tool. That means that only the owner tool should change those model elements. However this is just a rule of thumb, since certain operations on model elements may have consequences on other elements as e.g. cascading delete.

C. Model Versioning

The use case diagram in Figure 2 shows a high level view on the major use cases of a MoVE infrastructure. We can roughly group the use cases into model versioning use cases and change management use cases.

The actors are typically (human) stakeholders, that are in charge either to maintain the meta model as a meta model designer, or to maintain his/her partial model.

Besides human stakeholders there may also be special (automatised) processes that monitor the physical system and forward system data automatically into models. An example could be an application that monitors the state of all IT-applications running on a component and reports the data back into the enterprise model.

One major use case is Define/Update Model, where the model can be either a partial model or the meta model.

Updating a partial model may cause conflicts with the system model. Conflicts may result from classical versioning conflicts that are related to concurrent changes by different stakeholders. In this case classical conflict resolution techniques, e.g. UML compare[7], may be applied. Other conflicts may result from the fact that constraints C from the MoVE meta-model are violated. The hard option for handling constraint violations would be, not to allow the commit of the partial model into the system model unless the conflicts are resolved. Since a violation is not necessarily restricted to a single partial model, or requires the interaction of several stakeholders (e.g. a cascading delete crossing model boundaries), we will just notify the concerned stakeholders that commits the change about this violation.

Besides the system model, also the meta model may evolve over time. In this case mechanisms have to be provided to propagate changes in the meta model to the model layer. This is related to the large field of model migration, co-adaptation and co-evolution.

D. State Management

The second major functionality of a Living Models infrastructure is the change identification, propagation and notification.

Each class may have one or more distinguished attributes that represent the state of each instance. State transitions of such an instance are governed by an associated state machine. The set of state machines SM and the mapping m of the state machine to the distinguished attributes are defined in the MoVE meta model (MM, C, SM, m:SM→Attributes(MM)).

A very simplified example taken out of the health@net project is a business security objective instance that is associated with security requirements instances which again are associated to ModelElements’3 instances in the enterprise model (Figure 3). Each business security objective may be in one of the states (Figure 4).

3Unfortunately the health@net project decided to name the super class of all classes of the enterprise model, as processes, services, nodes, etc. ‘ModelElement’, which should not be mixed up with the concept of model element in Section IV-A.
**ADDED** meaning this business security objective is identified by somebody, but not yet evaluated.

**COMPLETE** meaning that all model elements instances in the enterprise model associated (indirectly by security requirements) with this security business objective are identified

**EVALUATED** the implementation of the security objective is evaluated.

The state machine in Figure 4 defines the state transitions of the business security objective subject to the state of related model elements in the system model. If a business security objective is newly added to the model, it must be in the initial state ADDED. The transition from ADDED to COMPLETE is only possible if all associated model elements are in the state COMPLETE. If security requirements are added and/or related model elements change their state back to ADDED, this change also resets the state of the business security objective back to ADDED. The (user-controlled) transition to EVALUATED is only possible, if all associated security requirements have the state EVALUATED.

Reflecting this on the use case diagram in Figure 2 the use-case “define/update model” can trigger the use-case change identification: The previous and the new version of the model are compared and the differences identified. If a state attribute value of an instance was changed, the corresponding state transition of the associated state machine is identified. I.e. we identify which state transition would have manifested in this state change. If no valid state transition can be identified, the new version is rejected.

In the positive case a state transition may emit an event that triggers further state transitions, which propagates state changes to other model element instances.

Finally, there must be the concept of “change notification”. This notification can be something quite simple as sending an email to the stakeholder that is responsible for the automatically changed model element, or as comfortable as an integration with a workflow engine that tracks those changes and manages a task list for the stakeholder.

V. CURRENT PROTOTYPE

We have implemented an initial prototype in order to study the usage scenarios of Living Models.

Currently it is based on a REST [25] interface to update and commit models and meta–models. Conflict resolution is based on EMF Compare [7]. State machine support is included based on SCXML [26].

In context of our case study (see III) we have implemented two adaptors, one for MagicDraw and another one for a proprietary application developed on top of a MySQL database. A somewhat arbitrary decision was, to use the UML meta model as a representational base for the partial models. This was the best choice for a research prototype, because a lot of UML–support is available in eclipse EMF. However the performance issues would not be acceptable in a serious application context.

VI. CONCLUSION

In this paper we have shown the major requirements and architectural issues of a Living Models infrastructure. Our intention is to provide an environment to help to keep models up–to–date and alive. The central issues are model versioning and change–driven model evolution. Especially the second issue needs excellent tool support in order to ensure a lively modelling culture. Manual checks would fail, because models and their interdependencies tend to become more and more complex.

The challenge is to bring the results of different model research areas together. This is a complex task, since these areas are still bleeding edge technologies, with all its problems as lack of documentation, low attention on interoperability, or missing scalability for larger models.

We are currently planning for the next version of the prototype that brings in additional functionality, as e.g.

- based on established (model) versioning tools
- support for meta model co–evolution
- defined interfaces for tool adapters
- defined interfaces for change notification

The goal is to provided a more scalable, pluggable and workable environment for a Living Models infrastructure.

REFERENCES


