Ten Principles for Living Models — A Manifesto of Change–Driven Software Engineering

Ruth Breu
University of Innsbruck
Institute of Computer Science
Techniker Strasse 21a, 6020 Innsbruck, Austria
phone: 0043/512/507-6114, fax: 0043/512/507-9871, mail: ruth.breu@uibk.ac.at

Abstract

The new generation of open networked IT systems poses particular challenges to software engineering due to their evolving nature and their high quality requirements. In particular, the management of service oriented systems requires the integration of perspectives from IT management, software engineering and systems operation and a systematic way to handle changes. In this paper we will present the core ideas of Living Models – a novel paradigm of model–based development, management and operation of evolving service oriented systems. A core concern of Living Models is to support the cooperation of stakeholders from IT management, software engineering and systems operation by providing appropriate model-based abstractions and the fostering of interdependencies. Based on this idea the running services together with their modelling environments constitute the basic unit of quality management and evolution. Living Models provides a coherent view of the quality status of the system (integrating the perspectives of all stakeholders) which evolves together with the running systems. This comes along with a software engineering process in which change is a first–class citizen.

Keywords

Service oriented systems, systems evolution, quality management, software engineering process.

1. Introduction

Agility, flexibility of business processes and efficient cooperation across organisational boundaries have emerged as important success factors of enterprises. Agile, flexible and cooperative business services require agile, flexible and cooperative IT services. Therefore in the recent years a broad range of technologies, standards and architectures has been developed. Most of these approaches follow the paradigm of **service oriented systems**. Basically a service oriented system consists of a set of independent stakeholders offering and calling services. Orchestration and choreography technologies allow the flexible composition of services to workflows [22], [16]. Arising application scenarios have demonstrated the power of service oriented systems. This ranges from the exchange of health related data among stakeholders in health care, cross–linking of traffic participants to new business models like SAAS (Software as a Service).

While major international efforts in industry and academia so far have focused on the development of standards, technologies and frameworks for realising service oriented systems only a minority of approaches deal with software engineering aspects. This contrasts with the challenges attached with the design and operation of service oriented systems.

First, service oriented systems in most cases are dynamically evolving systems. For instance, in a networked health care scenario this means that information exchange will start with a number of hospitals and practitioners and will be successively extended by new stakeholder instances, stakeholder types (e.g. pharmacies, laboratories), services and workflows. As a consequence the borderline between software engineering and operation of these systems at runtime almost disappears.

Second, due to the open nature of these systems complex quality properties like functional correctness, security and privacy of processed information play a major role for design and operation of service oriented systems. In the health care scenario, the correctness and integrity of patient–related data is of uttermost importance for the safety of human lives. In addition, many security requirements are imposed by legal regulations. In the health care scenario this includes complex access rules to patient related data and the ownership of the patient with respect to his/her data. In this respect many of these quality attributes have a strong dependency with IT management, e.g. concerning the compliance of systems. In addition many quality attributes are enforced by configuration at runtime, thus require the consideration of the runtime environment in the quality management process.

In the current stage of development it is commonly accepted that model based software development provides a valuable contribution to meet parts of these challenges. In particular, in the area of service composition corresponding products (e.g. [1], [2]) have found their way into practice. However, most existing frameworks focus on the construction of solutions and neglect quality support and the consideration of change as a first–class citizen. In this respect they are not yet capable to meet the challenges imposed by dynamically evolving systems and their need for systematic quality management.

In this paper we present the principles of the novel paradigm of **Living Models**. Living Models focuses on model based management, design and operation of dynamically evolving systems. The innovation of our approach is threefold. First, Living Models is based on the tight coupling of models and code. This comprises a bi–directional information flow from models to code and from code and the runtime system back to

This work was partially supported by the SecureChange (ICT-FET-231101) EU project.

the models. Second, Living Models supports the management of quality requirements fostering the cooperation between stakeholders in IT management, software engineering and systems operation. Third, Living Models is attached with a novel process model focusing on change and change propagation.

The name "Living Models" points to the tight coupling of the models with the running (=living) systems and to the support of the stakeholders' daily tasks at appropriate levels of abstraction. We developed this paradigm based on our experience of several years of work in the area of model driven software developments with a focus on security engineering and model based quality assurance [3], [10], [11], [8]. At the same time it is a starting point for the elaboration of methods and tools for the systematic development of high quality service oriented systems.

The remainder of this paper is structured as follows. In Section 2 we clarify the problem context and the core idea of Living Models. This is broken down into ten principles presented in Section 3. Finally Section 4 summarises related work and draws a conclusion.

2. Problem Statement and Core Idea

An example for the potential effects of a rather small change is the computer failure at Lufthansa in 2004. [1] The reason for the failure was the launch of a new computer programme, which lead to a software problem that brought the check-in system down. As a result of this failure 60 European flights were cancelled, affecting about 6,000 passengers. Also other partners in the airline's network were suffering delays as a consequence. This small example shows that the analysis of quality attributes (here: availability of the check–in system) in many cases requires the analysis of interdependencies across the layers ranging from IT management, software engineering to systems operation. This comprises the following tasks and roles.

IT Management is concerned with planning, management and analysis of IT services from the viewpoint of the business strategy and the business processes. This comprises responsibility for the recognition and evaluation of IT related risks and the compliance of the IT services. Typical roles within IT management are the CIO (chief information officer) and the CSO (chief security officer).

Software Engineering deals with the design, implementation, maintenance and evolution of IT services. The task of software engineers is to establish agreement on requirements, quality attributes and resources and to deliver IT services satisfying these requirements and quality attributes based on the available resources. Typical roles within software engineering are the requirements engineer, the software architect and the quality manager.

1. http://www.usatoday.com/travel/news/2004-09-24-lufthansa-cancellations_x.htm (accessed on 2009-07-22).

Systems Operation aims towards the reliable and secure operation of IT services. This comprises the deployment and configuration of IT services, their runtime monitoring and the recognition of runtime–related risks. Typical roles within systems operation are the system administrator and the platform responsible (e.g. for ERP applications).

In an evolving context the integrated view of the three domains together with a continuous quality management process is crucial (cf. Figure 1). In order to materialise this concept three major research challenges have to be met.

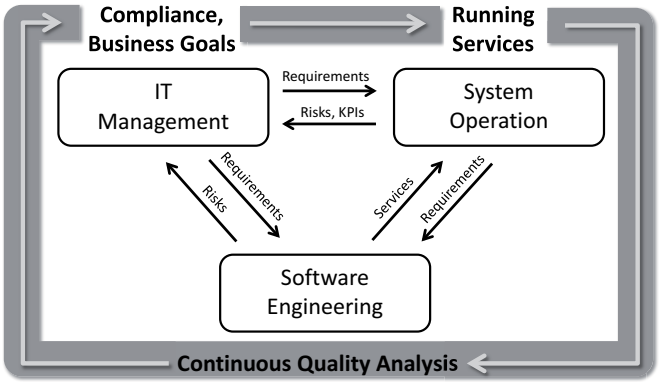

Fig. 1. An Integrated View of IT Management, Software Engineering and Systems Operation

RC1 – To provide a coherent view of the quality status of the system (integrating the perspectives of all stakeholders)

RC2 – To keep track of the quality status while the system evolves

RC3 – To support the collaboration of stakeholders for achieving the necessary quality level

Our core idea to meet these challenges is a rigorous use of models to bridge the gap between the domains and the related stakeholders. Figure 2 depicts the general concept of Living Models.

In a Living Models environment the basic unit of evolution constitutes the executable system together with its model–based views. Each view provides a working platform supporting specific stakeholders like the requirements engineer, the security officer or the tester. Each view records the current state of the system at an appropriate level of abstraction together with the current quality state. The quality state may be determined by qualitative attributes (evaluated by the responsible stakeholder) or by quantitative attributes (e.g. based on information collected in the runtime environment). Among the task of the stakeholders is to transform each quality attribute into its *target state* (e.g. to transform risks into the state where the current risk level is acceptable).

Changes in any of these views are propagated to the other views and stakeholder tasks are generated. The goal of these tasks is to transform the system again in its target state. For

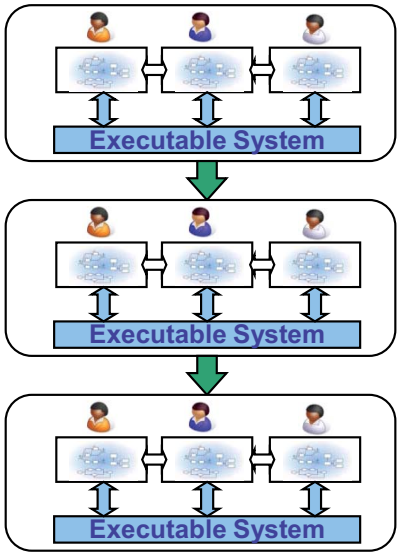

Fig. 2. A Living Models Environment

instance, the change of a legal requirement may require the security and risk analysis of relevant IT services, while the recognition of a new technical threat (e.g. change of a running service) may require the reanalysis of its business impact.

In this respect the entire system (= executable system together with its model views) strives to a state in which the quality attributes are in their target state. The prerequisite of this concept is the documentation of dependencies between model elements (including the code) which is in our opinion an indispensable foundation for an evolutionary approach.

There exist examples of environments supporting parts of the concepts of Living Models (or, seen it from the other side, the Living Models paradigm has been developed based on the success factors of these environments). As a first example, workflow management and EAI tools provide their users with full-fledged platforms abstracting from the workflow engines. Workflow engineers do not have to bother about the low–level issues of the workflow engine. The model–based environment supports their tasks to define workflows, to assure their quality and to monitor their runtime behaviour. A further successful application area are model–based testing environments (e.g. [20], [18], [19]). These environments are tightly connected with the executable system supporting a bi–directional information flow – calling the system's functions and transforming the results back to the modelling environment. However, also these success stories support aspects of system evolution and quality assurance only in an ad–hoc way.

3. Ten Principles for Living Models

In the following we will identify ten principles which we consider to be crucial for establishing a Living Models environment. As illustrative examples we will use models and case studies from our frameworks SECTET (model–based configuration of security architectures and ProSecO (collaborative security analysis and management) [10].

P1 – Stakeholder–Centric Modelling Environments

Living Models environments should be stakeholder–centric in the sense that they are targeted to specific stakeholders and their tasks. Most importantly this means that the stakeholders can operate on an appropriate abstraction level and lower levels are hidden to them.

As an example the CIO operates based on concepts like business processes, information objects and IT services for aligning IT landscapes with business goals and for software project planning. The system tester would like to formulate, execute and evaluate test cases at the level of the system requirements.

A Living Models environment does not necessarily have to provide a homogeneous modelling environment. More importantly all modelling environments have to contribute to a common system view (see P3).

The models in a Living Models environment provide the backbone for the stakeholders' work and collaboration. In order to support the tasks of a stakeholder properly the models may be enhanced by any kind of information. This may range from simple attributes like the cost of some model element (e.g. of a purchased software service) to service level agreements or emergency plans. This includes facilities to connect this information, like the support of cost calculations.

P2 – Close Coupling of Models and Code

In contexts where the running system is the main target any model and sophisticated model analysis is useless if the model does not reflect the running system. Therefore in an evolving environment the tight coupling of models and code is crucial. We talk of tight coupling if the code and the models are in consistent states and if there is some tool support to link models with code. This may comprise the following patterns.

- Models are used to generate code at design time
- Models are used to configure code at runtime
- Models are generated out of the code (e.g. architecture models)
- There is a tool–supported direct connection of models and code, e.g. test cases generated out of some requirements model are executed on the running system
- Model elements and code components are linked with each other and changes in either of them are propagated to the other end enforce change of the other

Combining P1 and P2 a Living Models environment consists of a set of modelling environments where each of these modelling environments supports the work of specific stakeholders at an appropriate level of abstraction and has a direct link to the executing system.

P3 – Bidirectional Information Flow between Models and Code

In order to provide a full-fledged working platform it is recommendable that information does not only flow from models to code and the running system, respectively, but also vice versa. As an example, workflow management systems feed back runtime information (number of running workflow instances, average completion time and the like) to the model level. This supports stakeholders in their task to assess and to improve workflow schemas.

In a more general view we conceive a Living Models environment to have sensors in the runtime environment. These sensors collect information which is brought back to the modelling environment and interpreted at the level of the model elements. Examples are information about usage frequency and duration of IT services which is brought back to the IT landscape model and supports the CIO in aligning IT services towards business goals.

In order to realise such sensors in a Living Models environment the following patterns can be used.

- generation of sensor software (observers and adapters) based on sensor definitions at model level
- injection of meta model information in the code, and
- code reflection in order to transmit runtime information to the code level.

P4 – Common System View

A prerequisite for a continuous quality management process and the collaboration of stakeholders is a common system view. In order to provide an appropriate level of preciseness and a basis for tool support the common system view is defined by a *System Meta Model* comprising meta model elements and their interrelationships.

The System Meta Model defines the backbone of the change management process. The meta model elements are the basic units of change and the associations between the meta model elements are the basis for change propagation. Figure 3 shows as an example a simplified System Meta Model. This meta model contains three *layers* for IT Management, Systems Operation and Software Engineering, respectively. Each of these three layers may consist of other layers, e.g. the Software Engineering layer comprises the sub-layers Requirements, SW Architecture and Code. Meta model elements range from business processes, information objects and organisational roles in the IT Management layer to (running) services and physical nodes in the Systems Operation layer and the (requirements view of a) service in the Requirements layer. The associations in the System Meta Model induce dependency chains from business processes and business objects to the technical layer (e.g. hosting servers). For more elaborated System Meta Models we refer to Enterprise Architecture models like the Zachman framework [23] or to the ontology of [7] in a service oriented context.

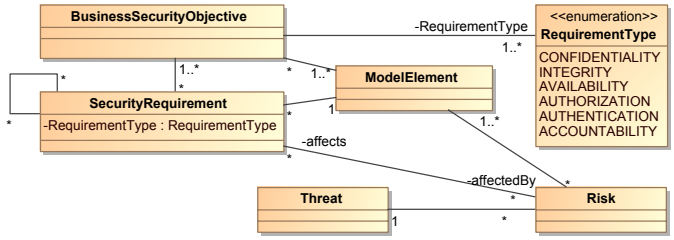

Fig. 4. Security Plug–In

While the System Meta Model describes the functional core concept of the quality analysis process extensions (so–called *plug–ins*) focus on non-functional aspects. For instance, Figure 4 shows a security plug–in. Each meta model element of Figure 3 is enhanced by security related information (e.g. security requirements, threats, risks). For detailed information we refer to [10]. Other plug–ins may describe a cost meta model or a performance meta model.

Referring to the usual notion of a model (like a business process model or a class diagram) each such model may be represented in an arbitrarily fine-grained way in the System Meta Model. For instance, a business process model may be represented entirely by the meta model element composite service or by its complete meta model (e.g. determined by the UML meta model). In our perception any information attached with a meta model can be integrated in the System Meta Model and its plug–ins and therefore be subject of traceability and change. This comprises code fragments (instances of the meta model elements Class, Component, cf. Figure 3) or textually described requirements (instances of the meta model elements Security Requirement, Threat, cf. Figure 4).

P5 – Persistence

The System Meta Model together with its plug–ins defines the schema for its instances (the so–called *System Models*). A System Model describes the current state of a system at all levels of abstraction (from business processes to physical nodes) including all relevant information (e.g. security risks, costs, quality state). As an example Figure 5 shows a fragment of a System Model as an instance of the System Meta Model of Figure 3 and the security plug-in of Figure 4. The depicted System Model has been developed in the course of a case study in which a networked health management system has been analysed [13].

System Models are the target of change and evolution. In order to support systematic analysis and reaction to change System Models have to be made persistent. In this spirit system evolution is described by a sequence of System Models. During the lifetime of the system not only the model elements but also the System Meta Model itself may be subject to change and evolution.

In addition, planning facilities should enable the analysis of different solutions (e.g. comparing different security safeguards concerning risk reduction and costs). This way,

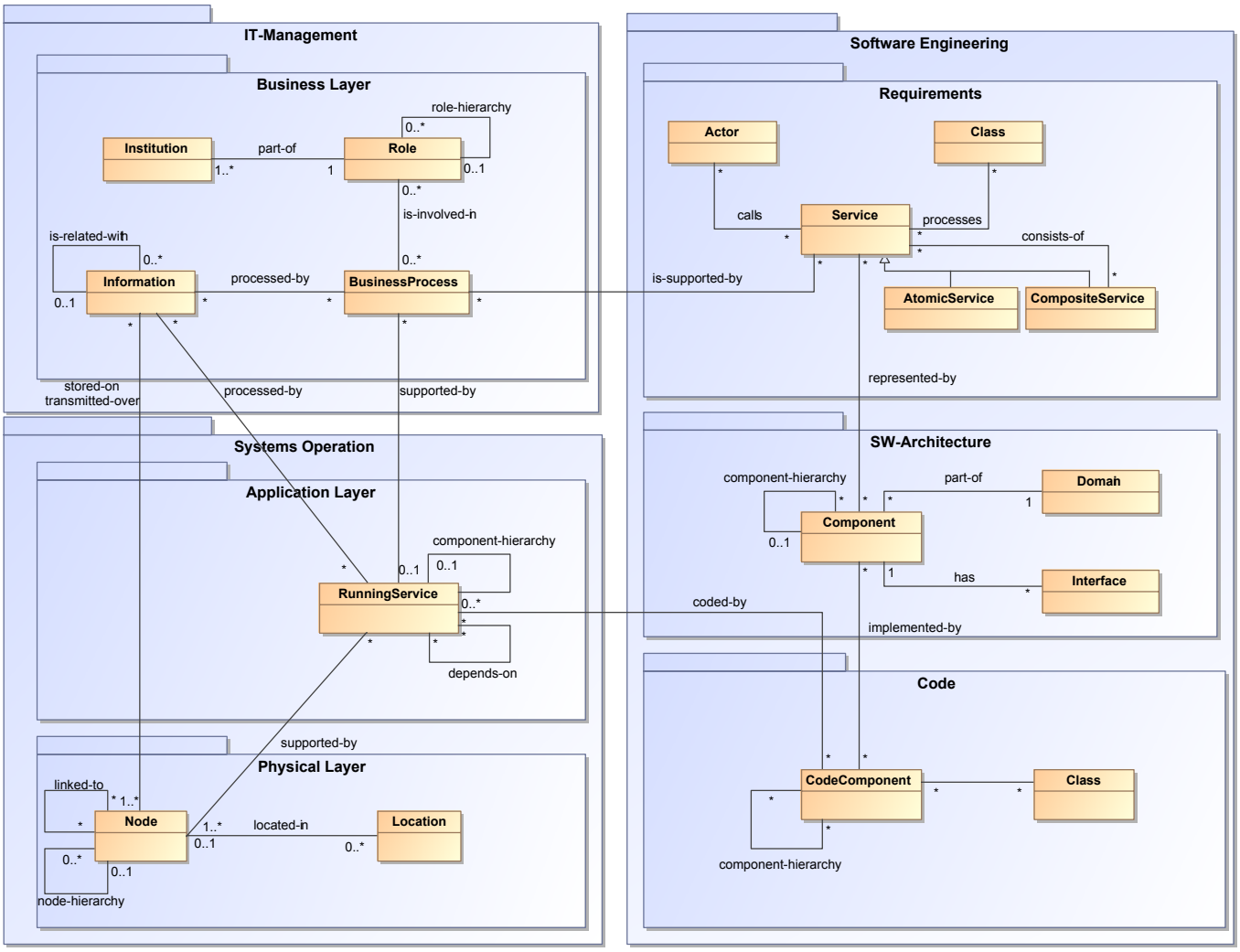

Fig. 3. Sample System Meta Model

the consideration of sequences of System Models has to be extended by the consideration of branches and System Model merging.

P6 – Information Consistency and Retrieval

State Models are complex networks of information describing the current state of the system. Constraints checking and information retrieval are two important services stakeholders have to be provided with. For instance, a CSO would like to check if every security threat at technical level is related with some security threat at business level (describing the business impact of the technical threat). Similarly, the software architect would like to check if each service at requirements level has a corresponding service at architecture level.

Information retrieval goes beyond such checks in delivering not only a Boolean value but any kind of information based on the System Model. The work of Buckl et al. [5] demonstrates that information visualisation plays an important role for information retrieval in System Models. Buckl et al. define a set of diagram types to visualise the connections between information at application and business layer. As an example, the process support map visualises which business processes are supported by which business applications in which organisational units.

P7 – Domains and Responsibilities

A System Model describes the current status of the entire system providing the basis for information traceability and cooperation among stakeholders. Each stakeholder operates on a subset of model elements in the System Model, e.g. being responsible for the quality assessment of these model elements.

More precisely, we attach the System Meta Model with a role model. Roles define the stakeholders (e.g. CIO, software architect, ERP applications responsible) and rights describe the actions the stakeholder is able to perform on the model elements, e.g. creation/modification of model elements or

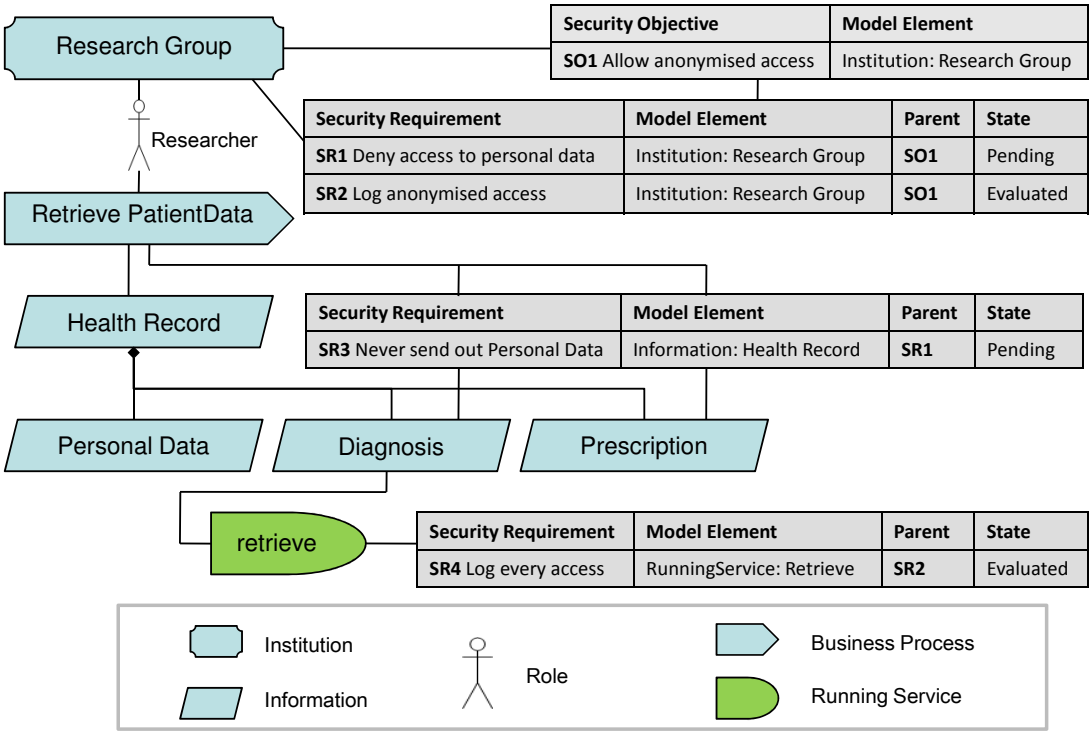

Fig. 5. Fragment of a Sample System Model

creation of certain events (cf. P9).

The set of model elements a stakeholder is responsible for is called his/her *domain*. For instance, the CIO may be responsible for the whole IT management layer, while the domain of platform responsibles is a range of model elements on the technical and physical layer.

P8 – Model Element States

Basically instances of the System Meta Model and its plug–ins are networks of model elements. Each such model element (e.g. a business process, an information object or a running service) can be attached with arbitrary information. This may range from a service level agreement, a model (e.g. an activity diagram) considered as black box in the change process, a key figure (e.g. the average number of running instances of a business process) or a *state*.

States play a particular role in the change–driven process since they determine the quality relevant milestones in the lifecycle of the model element. For instance, a requirement may be added, evaluated (i.e. was subject of a quality assessment activity) or implemented.

P9 – Change and Change Propagation

In our framework any change is perceived as an event which triggers consecutive steps of actions. Possible types of change events are the following.

- a time event (e.g. a threat analysis of the system has to be performed periodically)
- the change (modification/creation/deletion) of a model element in the System Model (e.g. the change of a compliance requirement or the change of a running service to a new release)
- events initiated by the stakeholders (so–called *action events*, e.g. the notification that a requirement has been validated manually or the notification that the set of security requirements is considered to be complete)

Change events are sent to the current System Model triggering change propagation and change handling according to the change–driven process described in P10.

P10 – Change–Driven Process

The software development process in a Living Models environment is driven by change events, the state of the model elements and their interrelationships with other model elements. More precisely, each change event a model element receives is attached with the following steps.

A. **State transition** – The change event may cause a state transition of the model element. For instance, the state of a requirement is changed from evaluated to added if the related model element (e.g. a software component) has been modified.

B. **Change propagation** – The state transition of the model element may trigger state transitions in related model elements according to given propagation rules. For instance,

the modification of a security requirement attached with a business process may cause state transitions in information objects and services supporting this business process. The propagation rules are specific to each meta model element.

C. **Modification of task list** – Each stakeholder is associated with a task list describing the pending action events of model elements he/she is responsible for. After each state transition new tasks may be pending and have to be added to the task list. Consequently fired action events (e.g. after the evaluation of a model element) are withdrawn from the task list.

The concept described above is materialised through (UML) state machines associated with each meta model element of the System Model and its plug–ins. As an example, Figure 6 shows the state diagram of the meta model element Security Requirement (cf. Figure 4).

The *states* of the state machines represent the major milestones in the lifetime of the respective model element. A security requirement may be in the state added (the security requirement has been identified and attached with a functional model element, e.g. a business process), complete (associated risks have been identified and declared to be complete at the current stage of development) or in the state evaluated (all associated risks have been evaluated).

The state evaluated is the *target state*, i.e. the responsible stakeholder has to take actions in order to transform the model element into this state. In this spirit a System Model can be attached with a set of green and red lights (= model elements in their target state/not yet in their target state, respectively).

State transitions in the state machines are triggered by events of the following kind.

- time events (e.g. triggering analysis actions to be performed periodically)
- conditions on the system state (e.g. in Figure 6 the state of a security requirement is changed from complete to evaluated if all associated risks are in state evaluated)
- action events initiated by the stakeholders (e.g. with the action event complete in Figure 6 the stakeholder declares the set of associated risks to be complete)
- change events caused by the modification/creation/deletion of some model element

Based on the concept of state machines the above steps A, B, C can be realised as follows.

A. **State transition** – A change event causes a state transition of a model element according to its associated state machine.

B. **Change propagation** – The state transition is observed by related model elements eventually causing transition of their states.

C. **Modification of task list** – After each state transition the task list of the associated stakeholders is automatically modified based on the state machines and the current system state.

Note that the change propagation step may cause a chain of state transitions across layers and responsibility domains of the stakeholders. In particular, dependencies between model elements of different responsibility domains support the cooperation of the related stakeholders. For instance, in this way the observation of a threat at technical level may be forwarded and interpreted across the layers triggering actions at business layer (e.g. if a re–evaluated business risk is getting too high).

The change propagation rules are specific to the meta model elements and may depend on the current state of the system. For instance, a change event may only have local effect if it is evaluated as unimportant in associated model elements (e.g., if a modified model element attribute stays within a tolerated range of values).

The stakeholders' task list corresponds to the sequence of activities in known process models (e.g. the sequences of activities in the waterfall model [17] or the V-Model XT [21] or the sequence of security related activities in security engineering processes like SDL [12] or CLASP [9]). While these process models can be supported in our framework (as we show in detail in the accompanying paper [15]) the change–based process offers two main advantages.

First, the change–based process fosters process steps on parts of the system (the "delta" of change). Second the quality state of the system is recorded, can be analysed at any point of time and can be tracked by tools. The price to be paid is the maintenance of model elements and their interdependencies. In our experience many companies have started to collect huge amounts of data (e.g. about IT landscapes, run–time properties) but are not yet able to use its huge analysis potential since date interconnection is missing. In this respect the Living Models paradigm provides a first step to exploit this potential. However, it is clear that still a lot of work has to be done. This concerns both the elaboration of patterns for the tool-supported maintenance of model elements and their interdependencies (reducing efforts and costs) and considerations of return of investment.

4. Conclusion

In the preceding sections we have identified three research challenges for evolving open systems with high quality requirements and have sketched a model–based approach to meet these challenges. The major aspects of the new Living Models paradigm are the systematic handling of changes at all levels of abstraction, continuous quality management and the integration of IT management, software engineering and systems operation.

Living Models has been built upon the ideas of model based software development and Enterprise Architecture Management ([23], [7], [5]). Software processes controlled by state machines can be found in specific areas and tools of software engineering, e.g. in requirements specification [14] and bug tracking [6]. Moreover, the UNICASE system [4] is a framework for global software engineering similarly based on the interconnection of model elements. However, UNICASE

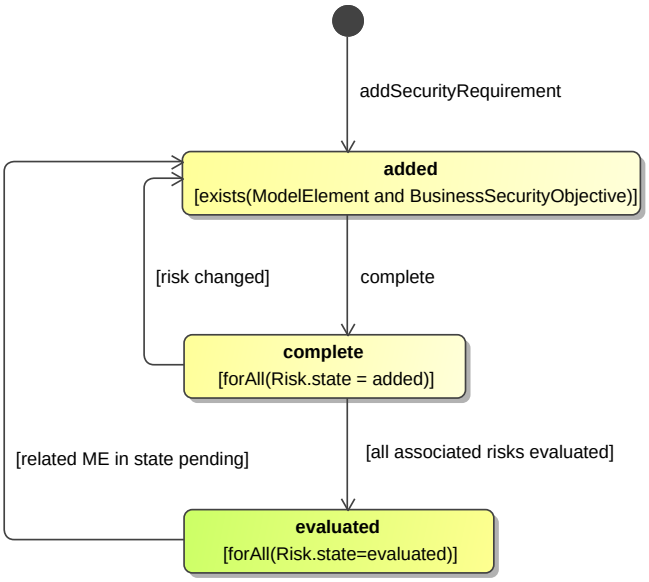

Fig. 6. State Machine of the Meta Model Element Security Requirement

does not support a change–based process and does not consider aspects of IT management and systems operation nor quality management.

On the one hand side the principles described in the preceding sections have been developed based on our experience in previous research projects conducted with several industry partners. In particular, many ideas of integrating software engineering and runtime aspects have been realised in the SECTET framework for model–driven configuration of security architectures [10]. The ProSecO security analysis framework [3] comprises a precursor of the change–driven process.

On the other hand side the described principles are a starting point for a whole bunch of activities. The change–based process is subject of research in the FP7–project SecureChange[2] with a focus on security aspects. Moreover, Quality Engineering has acquired the Excellence Centre Quality Engineering LaB in which several Living Models environments will be materialised in a four-years-program together with four industry partners. In particular, our next activities will be to develop tool support for the change–driven process, to explore the interface between IT management and software engineering and to develop a full–fledged Living Security environment. This is complemented by case studies which will be mainly conducted in the fields of networked health information systems and critical systems in telecommunication.

References

[1] Active Endpoints — ActiveVOS. http://www.active-endpoints.com.
[2] Oracle BPEL Process Manager. http://www.oracle.com/technology/products/ias/bpel/.
[3] R. Breu, M. Hafner, F. Innerhofer-Oberperner, and F. Wozak. Model-Driven Security Engineering of Service Oriented Systems. In *Information Systems and E-Business Technologies: 2nd International United Information System Conference, Uniscon 2008, Klagenfurt, Austria, April 22-25, 2008, Proceedings*, page 59. Springer, 2008.
[4] B. Bruegge, O. Creighton, J. Helming, and M. Kogel. Unicase — An Ecosystem for Unified Software Engineering Research Tools. In *Third IEEE International Conference on Global Software Engineering, ICGSE*, 2008.
[5] S. Buckl, A. Ernst, J. Lankes, F. Matthes, and C.M. Schweda. Enterprise Architecture Management Patterns – Exemplifying the Approach. In *The 12th IEEE International EDOC Conference (EDOC 2008)*, 2008.
[6] Bugzilla. http://www.bugzilla.org/.
[7] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus Voß, and Johannes Willkomm. A Method for Engineering a True Service-Oriented Architecture. In *ICEIS (3-2)*, pages 272–281, 2008.
[8] M. Felderer, P. Zech, F. Fiedler, J. Chimiak-Opoka, and R. Breu. Model-Driven System Testing of Service Oriented Systems. In *Proc. of the 9th International Conference on Quality Software*, 2009.
[9] Johan Gregoire, Koen Buyens, Bart De Win, Riccardo Scandariato, and Wouter Joosen. On the Secure Software Development Process: CLASP and SDL Compared. In *SESS '07: Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.
[10] M. Hafner and R. Breu. *Security Engineering for Service-Oriented Architectures*. Springer-Verlag New York Inc, 2008.
[11] M. Hafner, B. Weber, R. Breu, and A. Nowak. Model Driven Security for Inter-Organizational Workflows in E-Government. *Secure E–Government Web Services*, 1:233, 2006.
[12] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.
[13] Frank Innerhofer-Oberperfler, Markus Mitterer, Michael Hafner, and Ruth Breu. *Web Services Security Development and Architecture: Theoretical and Practical Issues*, chapter Security Analysis of Service Oriented Systems- A Methodical Approach and Case Study. IGI Global. In Press.
[14] In-Step. http://www.microtool.de/instep/en/.
[15] S. Löw and R. Breu. Strategies of Change–Based Software Engineering, 2009. In preparation.
[16] OASIS Standard. Web Services Business Process Execution Language Version 2.0 - OASIS Standard, April 2007. http://docs.oasis-open.org/wsbpel/2.0/.
[17] WW Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of the 9th International Conference on Software Engineering*, pages 328–338. IEEE Computer Society Press Los Alamitos, CA, USA, 1987.
[18] Smartesting. Test Designer, 07 2009.
[19] Testing Technologies. TTmodeler, 07 2009.
[20] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.
[21] V–Model XT. http://www.v-modell-xt.de/.
[22] W3C. Web Services Choreography Description Language Version 1.0, November 2005. http://www.w3.org/TR/ws-cdl-10/.
[23] John A. Zachman. A Framework for Information Systems Architecture. *IBM Systems Journal*, 38(2/3):454–470, 1999.

2. http://www.securechange.eu/