*Int J Software Informatics,* Vol.XX, No.XX, XXXXXX 2010, pp. 1–**??** E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2010 by Institute of Software, Chinese Academy of Sciences. All rights reserved. Tel: +86-10-62661040

# Living Models – Ten Principles for Change–Driven Software Engineering

Ruth Breu, Berthold Agreiter, Matthias Farwick, Michael Felderer,
Michael Hafner, Frank Innerhofer–Oberperfler
University of Innsbruck,
Institute of Computer Science,
Technikerstr. 21A,
6020 Innsbruck, Austria,
{ruth.breu, berthold.agreiter, matthias.farwick, michael.felderer,
m.hafner, frank.innerhofer–oberperfler}@uibk.ac.at

**Abstract** The new generation of collaborative IT systems poses great challenges to software engineering due to their evolving nature and their high quality requirements. In particular, the management of collaborative 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 a focus on 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. In this paper we will present ten core principles of Living Models together with three application scenarios.

**Key words:** Service oriented systems; systems evolution; quality management; software engineering process; evolution

**R. Breu, B. Agreiter, M. Farwick, M. Felderer, M. Hafner, F. Innerhofer–Oberperfler. Living Models – Ten Principles for Change–Driven Software Engineering.** *Int J Software Informatics*, 2010, XX(XX): 1–**??**. PAPER URL GOES HERE

## 1 Preface

I was among Manfred Broy's students when he got his first Professor's position at the University of Passau. So I can proudly state that Manfred Broy heavily influenced the long way from the first day of my studies (he was lecturer of "Introduction into Computer Science" in the first semester) to my habilitation and beyond.

† This work is sponsored by QE LaB – Living Models for Open Systems – FFG 822740, SecureChange EU grant number 231101, COSEMA – funded by the Tiroler Zukunftsstiftung and MATE – FWF project number P17380

His dedication to Software Engineering lead to the dedication to Software Engineering of a considerable amount of his students including me. I had the chance to take part in the development of a small research group at the University of Passau engaged in foundational research on formal methods to one of the largest research groups in Software Engineering at TU München being open minded to the manifold challenges of software development in practice.

Among the many visionary concepts Manfred Broy established in the research community I would like to mention the concept of system models as a foundational basis for modern multi–perspective modeling of systems. With our contribution I would like to demonstrate that also after 15 years of research on system models the comprehensive view of systems is still a hot topic.

Finally, I would like to take the opportunity and thank Manfred Broy for his great support of careers "with special needs". Many female (and also male) colleagues could tremendously benefit from his willingness to find flexible solutions. In my case this ranged from support to get scholarships during my family phase, untypical contracts (5 hours per week) to continuous encouragement.

## 2 Introduction

Agility, flexibility of business processes and efficient cooperation across organisational boundaries have emerged as important success factors of enterprises. Agile, flexible and collaborative business services require agile, flexible and collaborative IT services. Therefore, a broad range of technologies, standards and architectures has been developed in the recent years.

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 [30, 26]. 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. 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 core principles and application scenarios 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 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 many years of research in the area of model driven software development with a focus on security engineering and model based quality assurance [4, 17, 18, 11]. 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 3 we clarify the problem context and the core idea of Living Models. This is broken down into ten principles presented in Section 4. Section 5 demonstrates how these principles can be materialised in application scenarios in the domains of security engineering, enterprise architecture modeling and system testing. Finally, Section 6 summarises related work and draws a conclusion.

## 3 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

[1] http://www.usatoday.com/travel/news/2004-09-24-lufthansa-cancellations_x.htm (accessed on 2010-07-30).

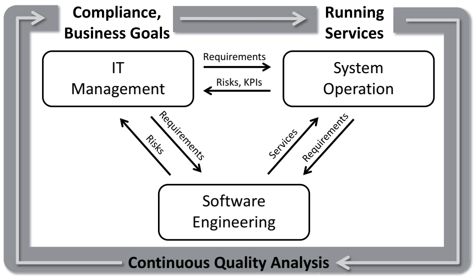


**Fig. 1.** An Integrated View of IT Management, Software Engineering and Systems Operation

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 CISO (chief information 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.

**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.

**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 and its requirements evolve

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

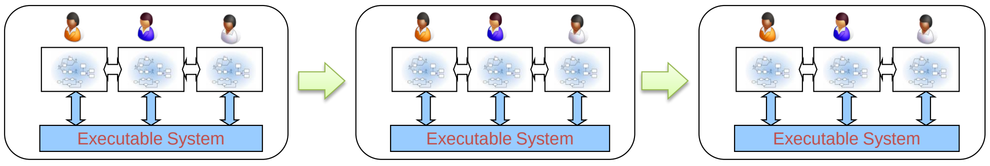


**Fig. 2.** A Living Models Environment

Our core idea to meet these challenges is a rigorous use of models to bridge the gap between the domains and the related stakeholders.

In a Living Models environment the basic unit of evolution constitutes the executable system together with its model–based views (cf. Figure 2). Each view provides a working platform supporting specific stakeholders like the requirements engineer, the security engineer 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). It is in the responsibility of the stakeholders to transform each quality attribute into its *target state* (e.g. to transform risks into the state where the current risk level is acceptable or to transform a component in a state where the number of passed test cases is above a certain threshold).

Changes in any of these views are propagated to the other views and stakeholder tasks are generated. Changes may have an effect on the quality state of a view and the goal of the tasks generated is to transform the system again in its target state. For 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.

## 4 Ten Principles for Living Models

In the following we will present ten principles which we consider to be crucial for establishing a Living Models environment. In Section 5 we will demonstrate how these principles are realised in different application scenarios and what benefits can be achieved.

Concerning the notion of models we accept any instance adhering to a Domain Specific Language (e.g. described by a MOF-compliant meta model [MOF]) as a model. For example, this comprises instances of (UML) class diagrams and state diagrams, but also enterprise models ([31, 28]) and even machine–executable code and semi–structured information like interlinked textual requirements descriptions.

*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 formulates, executes and evaluates 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 Running Systems*

In contexts where the running systems are 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 the runtime environment is crucial. We talk of tight coupling if the runtime environment and the models are in consistent states and if there is some tool support to link models with the running system. 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. A further example is 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 constitutes 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 traceability and change propagation.

Figure 3 and Figure 4 show a System Meta Model we use as reference model in our research platforms (cf. Section 5). For defining the System Meta Model we apply the following core concepts.

First, the System Meta Model comprises both functional and non–functional aspects of the system and attaches non–functional aspects to functional aspects. For instance, we do not talk of security requirements in general, but of security requirements of a business process or security requirements of a component. Thus, we associate the non–functional meta model element SecurityRequirement with the functional meta model elements BusinessProcess and Component, respectively. Figure 3 shows the functional part of the System Model (*Functional System Meta Model*), Figure 4 depicts the *Security Meta Model* as an example for non–functional aspects. Both diagrams are connected by the fact that the Security Meta Model refers to a meta model element FunctionalModelElement being superclass of each class in the Functional System Meta Model. Thus, the Security Meta Model enhances each functional model element by concepts such as a security objective, a security requirement or a security risk (for further information we refer to [17]). We call the Security Meta Model a *plug–in* of the Functional Meta Model. Examples of other plug–ins are a Test plug–in, a Cost plug–in or a Performance plug–in.

Second, the Functional System Meta Model defines the core concepts of the stakeholders' view on the system. The Functional System Meta Model is structured into three components for IT Management, Systems Operation and Software Engineering, respectively. Each of these components may comprise other components, e.g. the Software Engineering component contains the sub–components Requirements, SW Architecture and Code. The IT Management and Systems Operation component of our reference model is aligned with established Enterprise Architecture models like the Zachmann framework [31] or TOGAF [28]. They relate model elements from a business perspective (e.g. Information, BusinessProcess, Role) with model elements from an IT Application perspective (RunningService) and from a technical perspective (Node, Location), respectively. The Software Engineering component is structured into a requirements perspective (with the spanning concepts of a Service, an Actor and a Class, cf. [11]), a Software Architecture perspective (with logic components and interfaces as the core concepts) and the Code perspective (with code components and classes as the core concepts).

The associations in the System Meta Model induce dependency chains across the perspectives, e.g. from a business process to the server (Node) where its IT support (Running Service) is deployed.

Third, a System Meta Model defining the atomic meta model elements may be attached with a set of *views*. Each view represents a portion of the System Meta Model aggregating the meta model elements relevant to specific stakeholders. As an example, the Security Meta Model (cf. Figure 4) defines a Security Risk View defining the model elements relevant for the management of IT security risks. Figure 5 defines a partial view representing the mapping between the logical components of the SW Architecture and the code components.

Similar to the conventions of most modeling tools we also call the composition of meta model elements to a view a *diagram (type)*, e.g. the Security Risk Diagram defined in Figure 4 and the Architecture to Code Mapping Diagram defined in Figure 5.

Referring to the usual notion of a model or diagram (like a business process model or a class diagram) each such diagram 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 BusinessProcess or by its complete meta model (e.g. determined by the BPMN 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 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).

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.

In addition, planning facilities should enable the analysis of alternative solutions (e.g. comparing security safeguards within different architectural variants). This way,

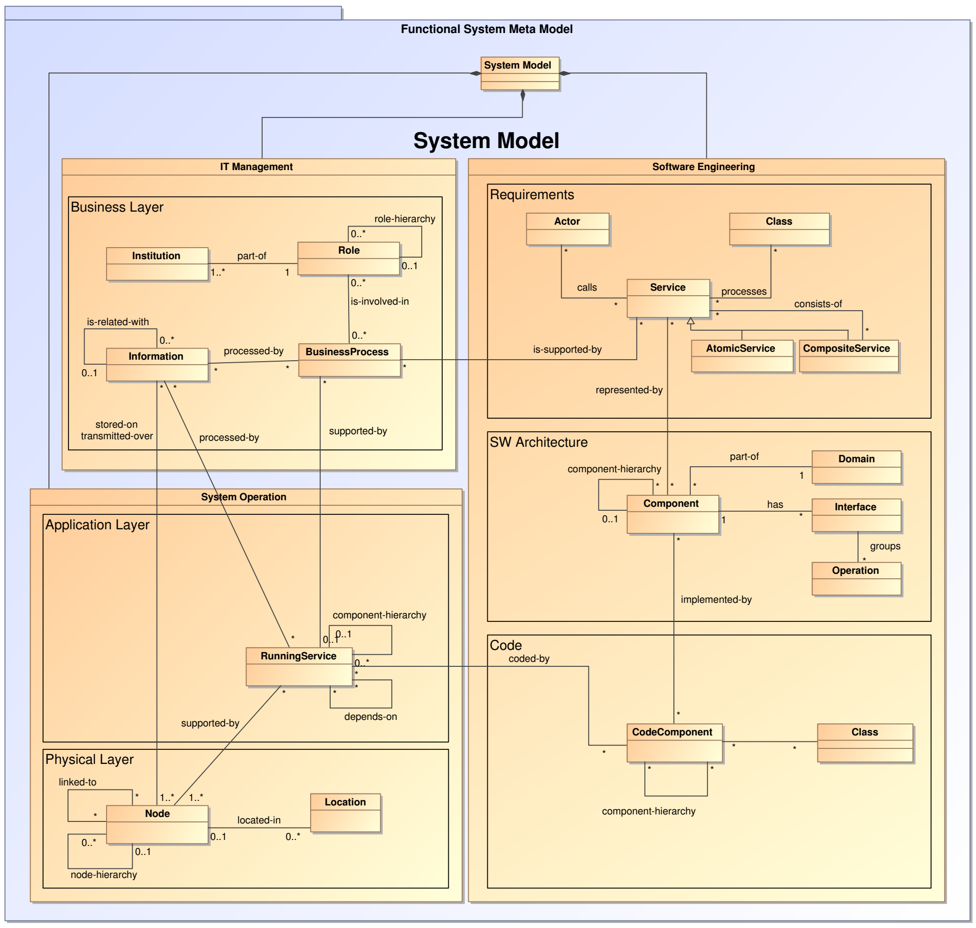


**Fig. 3.** Reference Functional System Meta Model

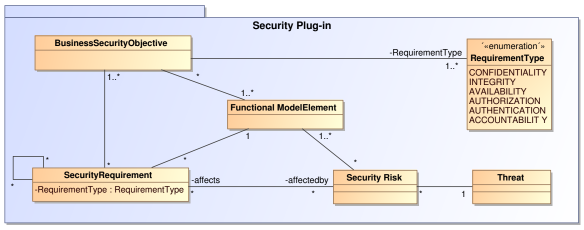


**Fig. 4.** Security Plug–In

the consideration of sequences of System Models has to be extended to the usual general view of versioning including branches and model mering.

Figure 6 extends the System Meta Model concept by a simple versioning concept based on a predecessor/successor relation. For clarification sample System Model instances are depicted.

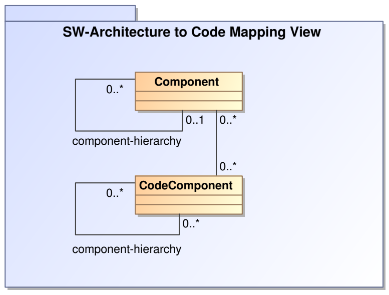


**Fig. 5.** Architecture to Code Mapping View

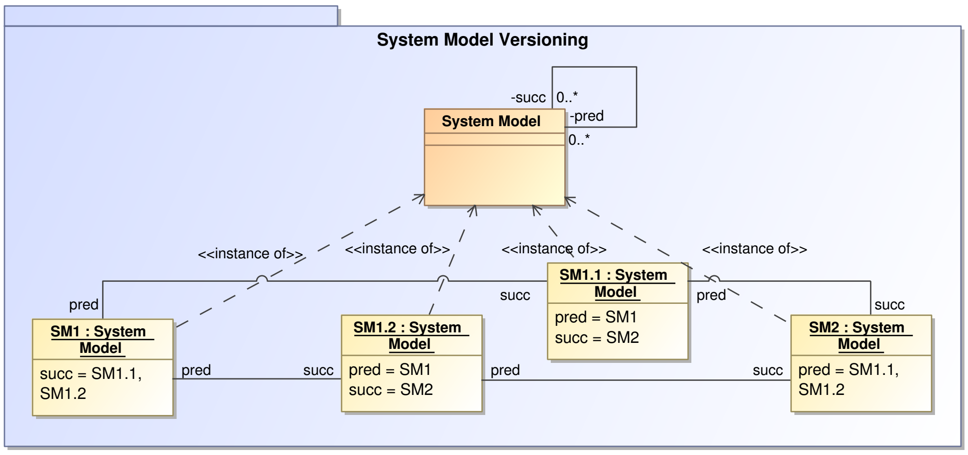


**Fig. 6.** System Model Versioning

During the lifetime of the system not only the System Models but also the System Meta Model may be subject to change and evolution. In this respect and in the spirit of the MOF language hierarchy [25] the System Meta Model itself has to be conceived as an instance of a meta model of a higher level. Concerning details on the requirements to model versioning in a Living Models environment we refer to [3]. For constructing and maintaining System Models the use of automated techniques is crucial. For instance, tools of static code analysis help to maintain interdependencies at code and software architecture level and model based testing tools help to link the requirements level with the code level.

*P6 – Information Consistency and Retrieval*

System 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 requirement 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 Hanschke [20] and Buckl et al. [6] demonstrates that information visualisation plays an important role for information retrieval in System Models. Both approaches 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 construction or 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 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 application responsibles is a range of model elements on the application and physical layer.

## *P8 – Model Element States*

In order to support the daily tasks of stakeholders it should be possible to attach each model element in a System Model (e.g. a business process, an information object or a running service) 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 Events*

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 requirements is considered to be complete)

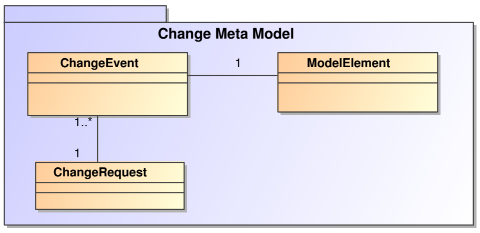


**Fig. 7.** Change Meta Model

Each change event is associated with a model element in the current System Model triggering change propagation and change handling according to the change–driven process described in P10.

In addition a *Change Request* conceptualizes the context of several change events (e.g. all change events related with the exchange of a running service). The notion of change requests provides a bridge to change management processes in IT Service Management (e.g. ITIL [23]). These processes describe the organisational handling of change, e.g. defining the necessary organisational structure (boards, responsibles) and process steps (e.g. prioritising and scheduling change requests). Figure 7 depicts the interrelationships between the change relevant concepts. Here we assume that every class of the System Meta Model is subclass of ModelElement.

*P10 – Change–Driven Process*

The support of a change driven software engineering process is an essential element in a Living Models environment. While conventional software engineering processes are controlled by sequences of activities a change driven process is controlled by change events, the current (lifetime) state of model elements and the interrelationships of model elements in the System Model. Change events trigger *change propagation* and a *change handling process*.

Change propagation determines which parts of the current System Model are affected by a change event. This may include the firing of new change events. Change handling is concerned with processing all relevant change events in order to finish the related change request.

More formally the change driven process is materialised through (UML) state machines attached to the meta model elements of the System Meta Model. The states of the state machines represent the major milestones in the lifetime of the respective model element and correspond to completed activities in conventional software engineering processes. Each state machine has one or several *target states* and it is in the responsibility of the associated stakeholder to transform model elements in their target state. In this spirit a System Model can be attached with a set of green and red lights, representing model elements in their target state or not yet in their target state, respectively.

As an example, Figure 8 shows the state machine of the meta model element SecurityRequirement. A security requirement can 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 target state evaluated (all associated risks have been evaluated). *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 8 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 8 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 state machines the change propagation and change handling process can be realised by the following three steps.

A. **Change Event** – A change event causes a state transition of the affected model element according to its associated state machine.
B. **Change Propagation** – The state transition may fire further change events sent to related model elements (or model elements may observe state transitions of related model elements). The change propagation rules realised in the state machines may vary from no propagation (the change is handled locally) to change propagation under certain conditions (e.g. only if a certain threshold is exceeded) or change propagation to selected related model elements, to change propagation to all related (or even all) model elements.
C. **Change Handling** – The change handling process is defined by the goal to transform all affected model elements of step B in their target state. In general this requires manual tasks (realised by action events) performed by the responsible stakeholders.

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 states in the state machines correspond to completed activities in known process models (e.g. the sequences of activities in the waterfall model [27] or the V-Model XT [29] or the sequence of security related activities in security engineering processes like SDL [21] or CLASP [16]). While these process models can be supported in our framework a change–driven process offers two main advantages.

First, the change–driven 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 this huge analysis potential since data 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

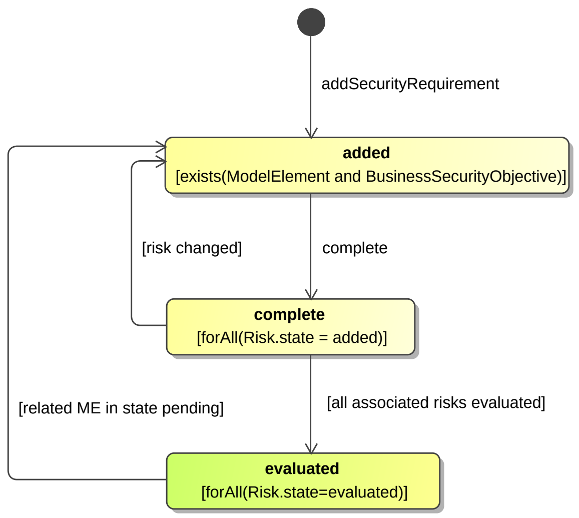

addSecurityRequirement
added
[exists(ModelElement and BusinessSecurityObjective)]
[risk changed]
complete
complete
[forAll(Risk.state = added)]
[related ME in state pending]
[all associated risks evaluated]
evaluated
[forAll(Risk.state=evaluated)]

**Fig. 8.** State Machine of the Meta Model Element SecurityRequirement

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.

## 5 Application Scenarios

In this section we demonstrate three application scenarios of the Living Models principles. Each of these application scenarios has been developed in our research projects in cooperation with industry partners.

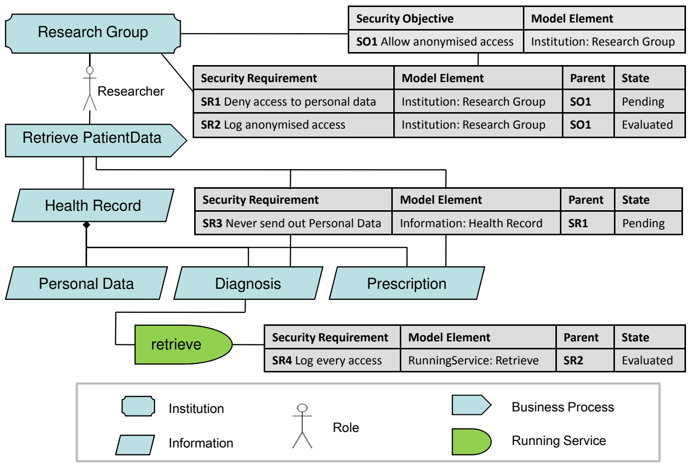
- SECTET/COSEMA is a framework for the model–based analysis of security requirements and risks (COSEMA) together with a platform for the model–based configuration of security services (SECTET). SECTET/COSEMA covers the full reference Functional System Meta Model (cf. Figure 3) and focuses on security as a quality attribute.
- Living Landscape Models is a tool–based method in the area of Enterprise Architecture Modeling. It covers the IT Management and Systems Operations domain of the Functional System Meta Model.
- Telling TestStories is a framework for model–based acceptance tests. It covers the Requirements and Code layer of the Software Engineering domain of the Functional System Meta Model.

### *5.1 The SECTET/COSEMA Living Security Models Environment*

#### *5.1.1 COSEMA*

In the project COSEMA[2] we work on a framework which supports a collaborative approach to information security management (ISM). The various activities related to

[2] http://www.cosema.org/



| Security Objective | Model Element |
|---|---|
| **SO1** Allow anonymised access | Institution: Research Group |

| Security Requirement | Model Element | Parent | State |
|---|---|---|---|
| **SR1** Deny access to personal data | Institution: Research Group | **SO1** | Pending |
| **SR2** Log anonymised access | Institution: Research Group | **SO1** | Evaluated |

| Security Requirement | Model Element | Parent | State |
|---|---|---|---|
| **SR3** Never send out Personal Data | Information: Health Record | **SR1** | Pending |

| Security Requirement | Model Element | Parent | State |
|---|---|---|---|
| **SR4** Log every access | RunningService: Retrieve | **SR2** | Evaluated |

**Fig. 9.** Fragment of a Sample System Model

ISM involve many different stakeholders in an organisation. Ranging from compliance and legal experts, to IT managers, network and system administrators, functional managers and last but not least end users each of these stakeholders play important roles in security management. To support a collaborative approach to the analysis of security requirements and risks COSEMA delivers a model–based framework which fosters the collaboration of these different stakeholders with their diverse backgrounds. Important aspects thereby are the support of communciation among these stakeholders and the provision of tailored perspectives reflecting the different levels of abstraction of the stakeholders.

The COSEMA approach is based on a three layered functional model of the enterprise comprised of business, application and phyiscal layer (cf. Fig. 3). This functional system model is extended with the security plug-in (cf. Fig. 4) to annotate the enterprise model with security concepts such as security objectives, requirements, risks and threats. Fig. 9 provides a schematic example of an enterprise model extended with security relevant information.

The COSEMA framework realizes the following principles of Living Security: Involving different stakeholders with diverse backgrounds it realizes *P1 – Stakeholder Centric Modelling Environments* to provide each of these stakeholders with tailored perspectives and representations of the models. Principle *P4 – Common System View* is realized to provide the means for traceability along dependency relations of the functional system model. In order to maintain consistency in the collaborative information security analysis principle *P6 – Information Consistency and Retrieval* plays an important role. Another important aspect to support the collaboration of differnent stakeholder is principle *P7 – Domains and Responsibilities* which allows to restrict access rights of the stakeholders to specific aspects of the enterprise model and security plug-in. Principle *P8 – Model Element States*, *P9 – Change events* are the basis for defining triggers and events that lead to new tasks that have to be carried out by a specific stakeholder.

### *5.1.2 The SECTET Framework*

The SECTET [17] framework supports business partners during the development and distributed management of decentralized peer-to-peer scenarios. Primarily developed for the realization of decentralized, security critical collaboration across domain boundaries – so-called inter-organizational workflows, it realizes a domain architecture aiming at the correct technical implementation of domain-level security requirements. It consists of three core components (cf. Figure 10): A. **Security Modeling**. The modeling component supports the collaborative specification of a scenario at an abstract level as a high-level *Platform Independent Model (PIM)).* The PIM consists of a Business Process Model complemented by Security Policies. The modeling component implements an intuitive domain specific language, which is rendered in a visual language based on UML2 for various modeling tools. The modeling occurs at a level of abstraction appropriate to bridge the gap between domain experts on one side and engineers on the other side, roles chiefly involved in two different phases of the engineering process – the requirements engineering and the design phase respectively.

B. **Code Generation & Model Transformation**. Business Process Models and Security Policies are mapped to a set of *Platform Independent Security Patterns (PISP)* leveraging abstract security services and protocols enforcing specified security policies. The engineer may choose among a number of alternative PISP reflecting vaious architectural options (e.g., brokered authentication vs direct authentication). The PISPs are then refined into *Platform Specific Security Protocols* of various granularity until they can be mapped into configuration code for the components of the target architecture. The layered approach is introduced in [24]. C. **Security as a Service – Reference Architecture**. The *Reference Architecture (RA)* transposes the model of Software as a Service to the security domain and thereby realizes *Security as a Service (SeAAS).* SeAAS resolves some of the conceptual and practical issues linked with the current practice to implement security functionality exclusively at the endpoint (e.g., significant processing burden on the endpoint, cumbersome maintenance and management of the distributed security infrastructures, lack of interoperability with external service requesters). But SeAAS also goes beyond the mere bundling of security functionality within one security domain as it realizes complex security requirements for decentralized processes involving two or more domains. We introduced SeAAS in [19].

The SECTET framework realizes several principles of Living Security. Primarily targeted at domain experts (*P1 – Stakeholder Centric Modelling Environments*), it relies on models to generate code for a target architecture (*P2 – Close Coupling of Models and Running Systems*). The collaboration of domain experts with security engineers and administrators is realized through a common view on the decentralized, security-critical process (*P4 – Common System View*). Although the framework's backbone consists of a common System Meta Model comprising meta model elements and their interrelationships, each stakeholder only operates on a subset of model elements through his customized modeling environment (*P7 – Domains and Responsibilities*). The framework supports to handle changing security requirements efficiently and effectively (*P9 – Change Events*). New requirements percolate from abstract models through all layers of abstractions (*P2 – Close Coupling of Models*

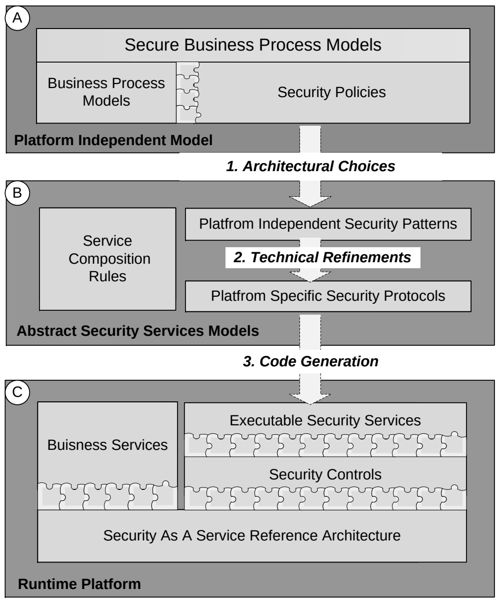

A
Secure Business Process Models
Business Process Models
Security Policies
Platform Independent Model
1. Architectural Choices
B
Service Composition Rules
Platfrom Independent Security Patterns
2. Technical Refinements
Platfrom Specific Security Protocols
Abstract Security Services Models
3. Code Generation
C
Buisness Services
Executable Security Services
Security Controls
Security As A Service Reference Architecture
Runtime Platform

**Fig. 10.** Sectet Framework for Model Driven Security

*and Running Systems*) and enable the stakeholders to take approapriate actions (*P10 – Change Driven Process*). The SECTET framework is currently being applied to use cases from the domotics industry and healthcare with our partners Telefonica I+D in the SecureChange project[3] and IT Icoserve GmbH in context of research related to Living Security Models of the QE-LaB Competence Centre [4].

### *5.2 Living IT–Landscape Models*

Enterprise Architecture Management (EAM), and in particular IT–landscape modelling try to model the IT and business elements of a company, in order to support the achievement of various goals in an enterprise. These goals include a business-aligned IT, achieving transparency over dependencies in the IT–landscape, being able to assess risks, secure regulatory compliance, and plan IT-landscape transformations in an enterprise as well as being able to analyse change impact. Conforming to our Reference Meta Model (cf. Fig. 3), typical EAM models consist of four layers. Among them are: A **Business Layer** describing the core business processes, business functions and products of an enterprise, a **Data Layer** describing the data items exchanged within the enterprise at an abstract level, an **Information System Layer** describing the enterprise information systems that support the business and their interconnections, and an **Infrastructure Layer** that describes the technical infrastructure which supports the information systems, e.g. servers and databases. In its simplest manifestation, EAM is conducted by modelling the enterprise-wide relations in a non–standardised way, for example on presentation slides. In its most sophisticated form, EAM is supported by

[3] http://www.securechange.eu/
[4] http://qe-lab.at/

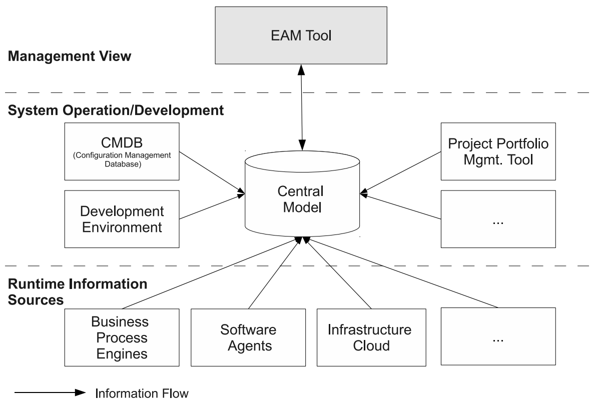


**Fig. 11.** The federated approach to Living Landscape Models

modelling tools[5][6], standardised notations and is accompanied with appropriate processes in the organisation. Both approaches, however, face a common problem. The elicitation of the necessary information to create the EAM models is time consuming, and due to rapid changes in enterprises, the information is often quickly outdated. This problem evokes a significant research question: How can enterprise architecture models, and IT–landscape models in particular, be automatically kept in–sync with the IT–landscape they represent, and how can information from different sources be correlated such that connections between model elements can be automatically inferred? Together with our industry partner iteratec GmbH[7] we tackle these questions by applying the principles of Living Models. For realising the Living Models principles in the context of enterprise architecture management we apply a federated approach to EAM as proposed by Fischer et al. [15] and extend it by making use of standardised interfaces and Semantic Web technology, i.e. logics. Figure 11 visualises this federated approach which we describe in more depth in [10]. The basic concept is that arbitrary information sources are connected to a centrally stored model. Each information source, such as project management tools, application servers and hardware monitors, contributes its information to this centralised system model. The communication facilities for this provision implement standardised interfaces [8]. When the central model receives data from a connected (runtime) information source, it adds this information to its semantic knowledge base. We then use rules to infer knowledge about relationships between model elements, and whether data from distinct information sources actually represent the same physical thing. However, this process still needs to be supervised by a human to avoid errors and to ensure desired quality. After the rules have been applied, the central model pushes the new information to the EAM tool. In particular we extended the open-source EAM tool iteraplan[8], which has been developed by our industry partner iteratec GmbH.

[5] http://www.iteraplan.de (accessed on 2010-08-09)
[6] http://www.ids-scheer.com/en/Software/ARIS_Software/ARIS_ArchiMate_Modeler/21980.html (accessed on 2010-08-09)
[7] http://www.iteratec.de
[8] http://www.iteraplan.de

Our approach and its prototypical realisation uses several of the Living Models principles. By keeping the EAM tool iteraplan in–sync with the real world, we provide a stakeholder centric view to the EAM practitioner and other involved actors (*P1 – Stakeholder Centric Modelling Environments*). Furthermore, information flows from running systems to the model and back (*P2 – Close Coupling of Models and Running Systems,P3 – Bidirectional Information Flow between Models and Code*). In our implementation, the Semantic Web knowledge base represents the common system view (*P4 – Common System View*) that also provides for persistence (*P5 – Persistence*). With rules and constraints we check the consistency of the aggregated knowledge, and, in addition, infer new knowledge (*P6 – Information Consistency and Retrieval*). The information flow in our system is driven by change events therefore *P9 – Change Events* and *P10 – Change Driven Process* are at the heart of our implementation.

Living Landscape Models allow for tight coupling of the EAM tool and other relevant information sources in an organisation. This directly tackles the synchronicity problem between EA models, resp. key figures relying thereon, and the infrastructure, and allows for making decisions with up-to-date information and quickly respond to changes in the environment.

### *5.3 Living Requirements with Telling TestStories*

Requirements testing methods for service oriented systems evaluate the system's compliance with its specified requirements and have to consider complex quality properties of systems and their dynamic evolvement. Model–driven testing approaches, i.e., the derivation of executable tests from test models by analogy to model–driven engineering are particularly suitable for requirements testing of service oriented systems because tests can be adapted easily to evolving requirements and complex quality properties can be expressed by models.

Telling TestStories (TTS) [13] is a tool–supported methodology for model–driven requirements testing of service oriented systems. TTS has been developed in cooperation with the German SME Softmethod GmbH[9] and applied to industrial case studies for functional requirements testing of a telecommunication application [14] and security requirements testing of a home networking application [12]. The core artefacts, their relationships and representation in TTS are depicted in Figure 12.

The **requirements model** contains the functional and non–functional requirements for system development and testing. Its structured part consists of a requirements hierarchy.

The **system model** describes the system structure and system behavior in a platform independent way. It is based on the notion of services. Each service is assigned to a requirement and to an executable service in the running system to guarantee traceability.

The **test model** defines the test requirements, the test data and the test scenarios as so called *Test Stories.* Test stories are controlled sequences of service operation invocations exemplifying the interaction of services and assertions for computing verdicts. Test stories may be generic in the sense that they do not contain concrete objects but variables which refer to test objects provided in tables. Test stories are validated,

[9] http://www.softmethod.de

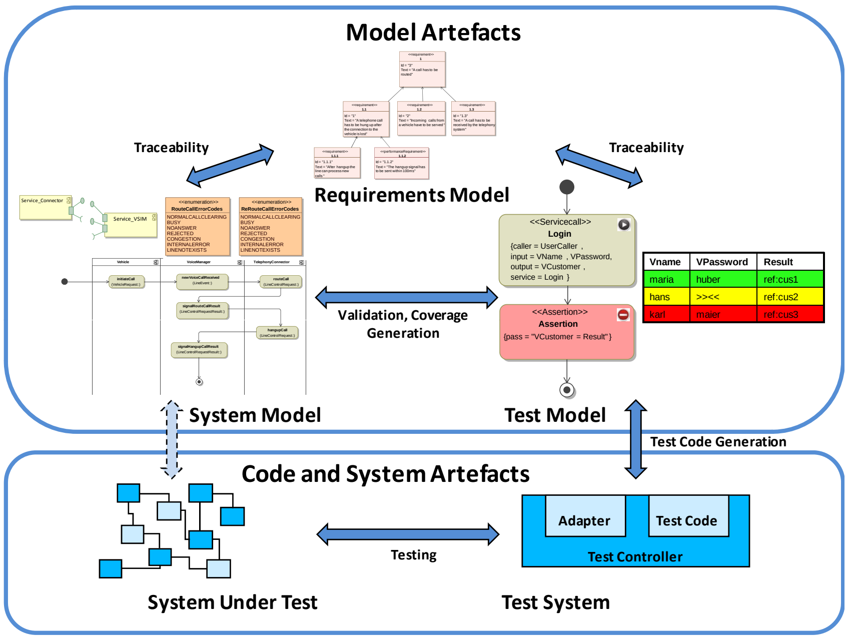


**Fig. 12.** Artefacts of Telling TestStories

checked for coverage or in specific cases generated from the system model. TTS supports classical test–driven development because with test models and adapters it is possible to achieve executable tests even before the system implementation has been finished. Test stories are linked to requirements and associated with services of the systems by service calls. Requirements are traceable to executable test stories and are therefore *Living Requirements*. The test controller, the adapter and the test code form the **test system**. The test controller executes the generated test code and accesses the system services via adapters. Our implementation of the test controller executes test code in Java. Adapters for each service are the link for traceability between the executable system, the test model and the requirements.

The **system under test** is a running service oriented system.

In TTS, traceability between all artefacts is guaranteed. Requirements, tests and services are associated on the model level via links between model elements. Each modeled service in the system is associated with an executable service in the system. The adapters link service calls in the model to executable services. Therefore every service invocation is traceable to a requirement.

TTS has been designed and implemented based on the principles of Living Models. Due to the traceability between all types of artefacts, TTS provides a close coupling of models and running systems (*P2 – Close Coupling of Models and Running Systems*) and a bidirectional information flow between models and code (*P3 – Bidirectional Information Flow between Models and Code*). For different stakeholders specific environments are provided (*P1 – Stakeholder Centric Modelling Environments*), e.g., a test report view for IT managers or a test model view for quality managers. TTS provides an integrated meta model for system, tests, and requirements under consideration of non–functional properties (*P4 – Common System View*) and checks for model consistency and coverage (*P6 – Information Consistency and Retrieval*). The remaining principles are considered in a TTS based approach to manage the evolution of test models by attaching state machines to all model elements. The test evolution framework then guarantees the consistent evolution of the system and its tests and uses the states of model elements to generate regression test suites.

## 6 Conclusion

In the preceding sections we have identified core research challenges for evolving open systems with high quality requirements and have presented ten principles which in our opinion are essentual 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 ([31], [9], [6]). Software processes controlled by state machines can be found in specific areas and tools of software engineering, e.g. in requirements specification [22] and bug tracking [7]. The UNICASE system [5] is a framework for global software engineering similarly based on the interconnection of model elements. However, UNICASE does not support a change–based process and does not consider aspects of IT management and systems operation nor quality management.

In Section 5 we demonstrated how the principles of Living Models are materialised in application oriented frameworks and what benefits can be achieved. These frameworks covered the fields of security engineering, enterprise architecture modeling and model–based acceptance tests.

There still remains a lot to be done. First, the Living Models paradigms in the presented frameworks still have to be elaborated in many facets. In addition, the systematic tool–supported management of interdependencies between model elements in the System Model is a major concern. Finally, we have started to conceptualise generic tool support for the change–driven software engineering process. This poses many challenges in direction of model versioning, merging models of various sources, co–evolution of meta models and models and supporting the state machine based change propagation and change handling process.

## References

[1] Active Endpoints — ActiveVOS. `http://www.active-endpoints.com`.

[2] Oracle BPEL Process Manager. `http://www.oracle.com/technology/products/ias/bpel/`.

[3] M. Breu, R. Breu, and Löw, S. Living on the MoVE: Towards an Architecture for a Living Models Infrastructure . In *Proc. 5th International Conference on Software Engineering Advances ICSEA 2010*, 2010.

[4] 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.

[5] 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.

[6] 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.

[7] Bugzilla. `http://www.bugzilla.org/`.

[8] DMTF. Configuration Management Database Federation (CMDBf), April 2010. `http://www.dmtf.org/standards/published\_documents/DSP0252\_1.0.1.pdf`.

[9] 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.

[10] M. Farwick, B. Agreiter, R. Breu, M. Häring, K. Voges, and I. Hanschke. Towards living landscape models: Automated integration of infrastructure cloud in enterprise architecture management. In *3rd International Conference on Cloud Computing - IEEE CLOUD 2010*. IEEE, 2010.

[11] 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.

[12] Michael Felderer, Berthold Agreiter, and Ruth Breu. Security Testing by Telling TestStories. In *Modellierung 2010*, 2010.

[13] Michael Felderer, Joanna Chimiak-Opoka, and Ruth Breu. Model–driven System Testing of Service–oriented Systems. In *Proceedings of the 12th International Conference on Enterprise Information Systems*, 2010.

[14] Michael Felderer, Philipp Zech, Frank Fiedler, Joanna Chimiak-Opoka, and Ruth Breu. Model-driven System Testing of a Telephony Connector with Telling Test Stories. In *Software Quality Engineering – Proceedings of the CONQUEST 2009*, 2009.

[15] R. Fischer, S. Aier, and R. Winter. A federated approach to enterprise architecture model maintenance. *Enterprise Modelling and Information Systems Architectures*, 2(2):14–22, 2007.

[16] 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.

[17] M. Hafner and R. Breu. *Security Engineering for Service-Oriented Architectures*. Springer-Verlag New York Inc, 2008.

[18] 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.

[19] Michael Hafner, Mukhtiar Memon, and Ruth Breu. SeAAS - A Reference Architecture for Security Services in SOA. *Journal of Univ. Computer Science*, pages 2916–2936, 2010.

[20] I. Hanschke. *Strategic IT Management: A Toolkit for Enterprise Architecture Management*. Springer, 2010.

[21] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.

[22] In-Step. `http://www.microtool.de/instep/en/`.

[23] ITIL – IT Infrastructure Library. `http://www.itil.org/`.

[24] M. Memon, M. Hafner, and R. Breu. SECTISSIMO: A Platform-Independent Framework for Security Services. In *ModSec '08: MODELS 2008*, Toulouse, France, 2008.

[25] OMG's MetaObject Facility. http://www.omg.org/mof/.

[26] OASIS Standard. Web Services Business Process Execution Language Version 2.0 - OASIS Standard, April 2007. http://docs.oasis-open.org/wsbpel/2.0/.

[27] 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.

[28] TOGAF – The Open Group Architecture Framework. http://www.opengroup.org/togaf/.

[29] V–Model XT. http://www.v-modell-xt.de/.

[30] W3C. Web Services Choreography Description Language Version 1.0, November 2005. http://www.w3.org/TR/ws-cdl-10/.

[31] John A. Zachman. A Framework for Information Systems Architecture. *IBM Systems Journal*, 38(2/3):454–470, 1999.