D2.3 An Integrated Security Process for Lifelong Adaptable Systems

Christian Connert (UIB), Simon Forster (UIB), Michael Hafner (UIB), Frank Innerhofer-Oberperfler (UIB), Philipp Kalb (UIB), Basel Katt (UIB), Sarah Löw (UIB), Stéphane Paul (THA)

Document information

Document Number	D2.3
Document Title	An Integrated Security Process for Lifelong Adaptable Systems
Version	1.0
Status	Final
Work Package	WP 2
Deliverable Type	Report
Contractual Date of Delivery	31 January 2012
Actual Date of Delivery	31 January 2012
Responsible Unit	UIB
Contributors	UIB, THA, SIN
Keyword List	
Dissemination level	PU

Document change record

Version	Date	Status	Author (Unit)	Description
0.01	06.02.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Outline
0.02	17.02.2011	Draft	Christian Connert (UIB)	HOMES
0.03	22.02.2011	Draft	Christian Connert (UIB)	Performance Study
0.04	09.03.2011	Draft	Christian Connert (UIB)	Extended and updated Performance Study
0.05	28.03.2011	Draft	Christian Connert (UIB)	Updated / formatted references (figure, etc)
0.06	28.03.2011	Draft	Christian Connert (UIB)	Updated figures
0.07	29.03.2011	Draft	Christian Connert (UIB)	Updated Performance Study
0.08	13.04.2011	Draft	Michael Hafner (UIB)	Revised Chapter 3
0.09	14.04.2011	Draft	Simon Forster (UIB)	MDS
0.10	19.05.2011	Draft	Simon Forster (UIB)	Revised MDS
0.11	30.05.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Streamlining
0.12	28.06.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Restructuring outline to fit in 50 pages limit
0.13	05.07.2011	Draft	Stéphane Paul (THA)	Minor update of the structure of Section 2 and Section 9.
0.14	11.08.2011	Draft	Stéphane Paul (THA)	Initial contribution to Section 2.

0.15	14.09.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Restructured according to conf call
0.16	15.10.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Draft of Sections 2, 3
0.17	23.11.2011	Draft	Sarah Löw, Philipp Kalb (UIB)	Integration of MoVE chapter
0.18	24.11.2011	Draft	Philipp Kalb (UIB)	Client side extension of MoVE
0.19	24.11.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Chapter 5 Evaluation and impact
0.20	25.11.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Added process description
0.21	12.12.2011	Draft	Philipp Kalb, Sarah Löw (UIB)	Changes in the MoVE section
0.22	13.12.2011	Draft	Basel Katt (UIB)	Changes in the SeAAS-MDS section
0.23	15.12.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	Final changes and integrations
0.24	19.12.2011	Draft	Frank Innerhofer-Oberperfler (UIB)	First version ready for scientific review
0.25	10.01.2012	Draft	Michela Angeli (UNITN)	First quality check completed – minor remarks
0.26	01.01.2012	Draft	Bjørnar Solhaug (SIN)	Scientific review completed – minor remarks
0.27	02.01.2012	Draft	Frank Innerhofer-Oberperfler (UIB)	Changes to address scientific review comments and quality check remarks
0.28	16.01.2012	Draft	Stéphane Paul (THA)	Second scientific review completed – minor remarks
0.29	19.01.2012	Draft	Michela Angeli (UNITN)	Second quality check completed – minor remarks

0.30	23.01.2012	Draft	Philipp Kalb, Sarah Löw (UIB)	Addes state of the art on tools in Section 3
0.31	23.01.2012	Draft	Frank Innerhofer-Oberperfler (UIB)	Final changes to address second scientific review and second quality check
1.0	24.01.2012	Final	Frank Innerhofer-Oberperfler (UIB)	Final version

Executive summary

The present deliverable reports on the research results of Work Package 2 in Year 3. In this deliverable the main focus is on the results of the finalization of the prototypical tool implementation. In addition to the written report Deliverable 2.3 comprises the documented software concerning the MoVE tool and the model driven security interface for the security-as-a-service-architecture (SeAAS-MDS).

In this deliverable we describe how the various tools developed in the SecureChange project map on the Integrated SecureChange Process which was presented in Year 2 in the deliverable D2.2. We then outline a specific process and tooling implementation on the basis of the HOMES case study using MoVE as the backend. In the HOMES scenario we connect three different tools with MoVE to outline how it can support collaboration and a change-driven engineering process.

A main aspect of this deliverable is the description of the MoVE tool. The MoVE tool stands for Model Versioning and Evolution and is a tailor-made model repository to support Living Models and change-driven processes. We provide a short summary of the objectives, the intended features and the architecture which were described in more detail already in D2.2. In this deliverable we focus particularly on the extendibility and configuration of the MoVE tool to outline how it can be used to support virtually any proprietary process and tool.

The other main aspect is the description of the model driven security interface for the Security-as-as-Service Architecture (SeAAS-MDS). We provide a short summary of and overview of the framework by outlining the input models and respective output policies and the transformation process. Then we focus on the prototypical tool itself, in particular on its graphical user interface which is presented in a step-by-step walkthrough. Another important aspect of the model driven security interface is its extendibility which is also addressed in this deliverable.

The deliverable concludes with a summary of the evaluation feedback we collected. We applied the MoVE tool on both the ATM case study for supporting a fine-grained change-driven process and the HOMES case study for supporting a coarse-grained change-driven process. In addition the ATM practitioners evaluated the methodology of change-driven engineering process. The Security-as-a-Service-Architecture (SeAAS) was integrated in the HOMES gateway. In addition we conducted performance experiments to evaluate the performance of the SeAAS approach in comparison to other approaches.

We collected valuable feedback from the practitioners and partly addressed their requests during the finalization of the prototypical tool development. In terms of impact we have published the research results of Work Package 2 in a number of international software engineering conferences and journals. The MoVE tool has already achieved positive impact in the research and industrial communities. The MoVE tool will be used as an infrastructure to integrate different artefacts and to support a change-driven process to be developed in the FP7 project PoSecCo. In addition three industrial partners showed strong interested in the MoVE tool and are currently evaluating the feasibility of integrating it with their tools.

Index

1 Introduction

In the first year Work Package 2 focused on the aspects of security engineering processes and security architectures for evolving systems (cf. D.2.1 [30]). In particular we presented the Living Security Engineering Process as the first fully change driven security engineering process. In Year 2 we started tool implementation and extended our work on change driven security engineering to provide an integrated process methodology to connect the results of the various Work Packages of the SecureChange project. This Integrated SecureChange Process was developed as a light-weight variant of a change-driven security engineering process.

In the third year of the SecureChange project Work Package 2 focused on the finalization of the prototypical tool development and the validation and evaluation of our results. In terms of tool development major efforts were invested in completing the MoVE tool implementation. MoVE, the shortcut of Model Versioning and Evolution, is a tailor-made model repository to support Living Models and change-driven processes. The MoVE tool has also been equipped with several adaptors which allow the direct import and export of models in various formats (XMI, Risk model in spreadsheet representation, and a model driven configuration interface for SeAAS). In relation to the Security-as-a-Service-Architecture (SeAAS) a model driven interface to support a high-level configuration of the security services was implemented (SeAAS-MDS).

In terms of validation and evaluation we worked on two different industrial case studies, the ATM and the HOMES case study. In terms of tool development major efforts were invested in completing the MoVE tool implementation. Subject to evaluation and validation on the ATM case study were on one hand the application of the change-driven security engineering methodology. On the other hand we have applied the MoVE tool on the ATM case study and exercised a live demonstration of the tool with industrial experts from the ATM domain. In terms of the HOMES case study we have integrated the Security-as-a-Service-Architecture (SeAAS) in the HOMES gateway and delivered this prototypical implementation to our industrial partner for evaluation.

Overview of this document

In Section 2 we describe the tool support for the Integrated SecureChange Process. One aspect is the provision of a general overview of how the various tools developed in the overall SecureChange project can be mapped on the Integrated SecureChange Process. The other aspect is the description of a specific process and tool implementation on the basis of the HOMES case study. We outline how three different tools are orchestrated in an integrated process using the MoVE tool as a backend.

In Section 3 we focus on the results of the MoVE tool implementation. The requirements and the overall architecture have already been documented in Year 2 (cf. D.2.2 [31]). In this deliverable we shortly summarize the main objective and core features of the MoVE tool. Particular emphasis is put on how the MoVE tool can be extended and configured to support any industrial process and proprietary tool. The description of a concrete extension and configuration is aligned to the demonstration of the HOMES case study.

In Section 4 we describe the model-driven configuration interface for the Security-as-a-Service-Architecture (SeAAS). This section contains in the first part a description of the overall framework, including the input and output models and the transformations needed to adapt the configuration. The second part is dedicated to the graphical user interface and to the extensibility of the tool.

Section 5 describes the feedback collected during the validation and evaluation exercises with our industrial partners and outlines how the feedback impacted the research and the prototypical tools of Work Package 2. This section contains also a short summary of the impact of the research results developed in Work Package 2. The Deliverable ends with a Conclusion in Section 6, a short Glossary in Section 7 and the Bibliography in Section 8.

In addition to this printed report Deliverable 2.3 also comprises the documented software of the MoVE tool and the model driven interface for the configuration of security services (SeAAS-MDS).

2 Tool Support for the Integrated Security Process for Lifelong Adaptable Systems

In this Section we sketch the tool support for the Integrated Security Process for Lifelong Adaptable Systems. The Integrated SecureChange Process was developed in Year 2 as a light-weight change driven security engineering process. That way the Integrated SecureChange Process allows the integration of different methodological approaches developed within the overall project. In this Section we will present how all the tools developed within the project map to the Integrated SecureChange Process.

2.1 Tool roadmap

This section describes how the various tools developed in the SecureChange project contribute to the Integrated Security Process. Figure 1 outlines the different tools with regard to the models that serve as input or output and the main activities that are executed. Most of the tools support a specific methodology developed within the project. Several of these tools are built on the basis of the Eclipse platform. This means most of the tools have a common technological platform which provides a basis for integration.

Two tools are generic in the sense that they are methodology-agnostic. One of these tools is EMF-IncQuery which is a very efficient query language and implementation. The other tool is the MoVE tool developed within Work Package 2. The MoVE tool supports the instantiation of the Integrated SecureChange Process. It is generic in the sense that it allows modeling any type of process and is open to connect any type of tool via specific adaptors. In Chapter 3 we describe in detail how the MoVE tool can be configured to support arbitrary processes and tools.

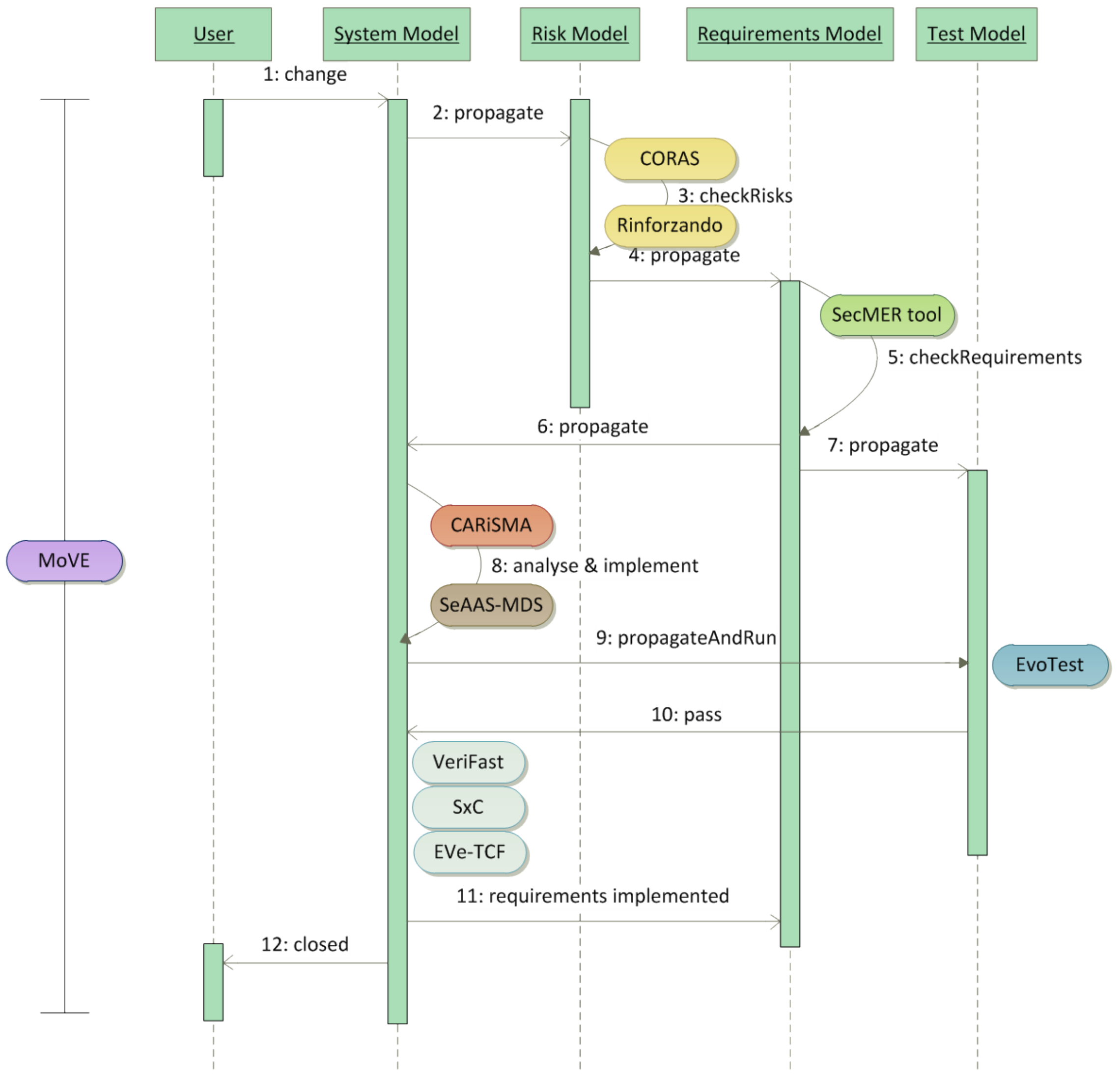


Figure 1 SecureChange tools mapped on the Integrated Process

2.2 The process of defining the Integrated Process

In many organizations there are existing industrial proprietary processes and tools which are in use. The Integrated Process can be adapted to virtually any proprietary processes and tools in place. In the following the main issues which have to be addressed in the process of redefining or adapting the Integrated Process are shortly described. The description supposes that the Integrated Process is supported by the MoVE tool which will be described in more detail in Section 3.

The first step is the analysis of any existing process descriptions, either by studying process documentations, by walkthroughs or through workshops with the relevant stakeholders. The main goal is to achieve an understanding of the dynamics of the current processes. The challenge in this task is the transition from an activity oriented view to a change-driven view.

A decision on the level of granularity has to be made. Should the integration be based on the states of entire models or on the states of single model elements? An example for a granularity based on entire models is the Integrated SecureChange Process. An example for a fine-grained integration based on the state of model elements is the change-driven security engineering process which was applied in D2.2 on the ATM case study (cf. [31]).

During the transition of an activity oriented view to a change-driven view it is crucial to gain an understanding of which activities are performed on which models or model elements. Related to this aspect are also the state changes that reflect the lifecycle of these model elements.

Core aspects of a change-driven engineering process are the change events which trigger activities. To correctly define these change events we need to have a clear picture on the links between the models and the model elements. Based on these links we then need to ask when and how change events can be processed. What underlying rules have to be fulfilled before certain actions can be fired?

The answers to the above outlined questions result in an initial design of a draft for the state machines. These state machines have not yet transitions and rules defined; they just highlight the major lifecycle phases of the artifacts which are processed.

Several feedback loops have to be run together with process owners or stakeholders to improve and correct the initial design of state machines. Once the state machines have reached the expected quality and correctly outline the artifact states at the right level of granularity then the informal design of transition rules begins.

These transition rules define when an artifact changes from one state to another and when changes in other state machines are triggered. The initial description of these rules is informal and textual to have a basis for the discussion with the stakeholders. Once the informal description of the transition rules is confirmed correct, the rules are formalized using the Object Constraint Language (OCL).

Using specialized tools (e.g. SQUAM[1]) the correctness of these OCL rules can be checked. Again several feedback loops might be required before the desired quality and correctness of formal OCL rules for the transitions is reached. At the end of the process also the state machines are formalized using SCXML. Both the formalized state charts defined in SCXML and the formal transition rules defined as OCL can be interpreted by the MoVE tool to support the synchronization of different tools and processes.

In the SecureChange project we have applied the process of defining an Integrated Process for a set of tools of our industrial partner Thales. Thales provided us with an initial process description of a system modeling tool (SMS) and the risk modeling tool Rinforzando. We have also been given consistency rules which define relations between the system and the risk model. The resulting state machines which reflect the

[1] http://www.squam.info/

lifecycle of the artifacts processed by the system analysis and risk analysis processes of Thales are documented in the deliverable D4.4b.

2.3 A specific process and tool implementation: The HOMES case study

In this Section we outline how the MoVE tool can be used to implement and support the Integrated Process by orchestrating and connecting different tools. The basis for this demonstration is the change scenario which was identified in the HOMES case study and described in deliverable D2.2, Section 7.1 (cf. D.2.2 [31]). In this deliverable a concrete tool based instantiation of the Integrated SecureChange Process will be described. For a detailed introduction to the HOMES case the reader is referred to deliverable D2.2, Section 7.1 (cf. [31]).

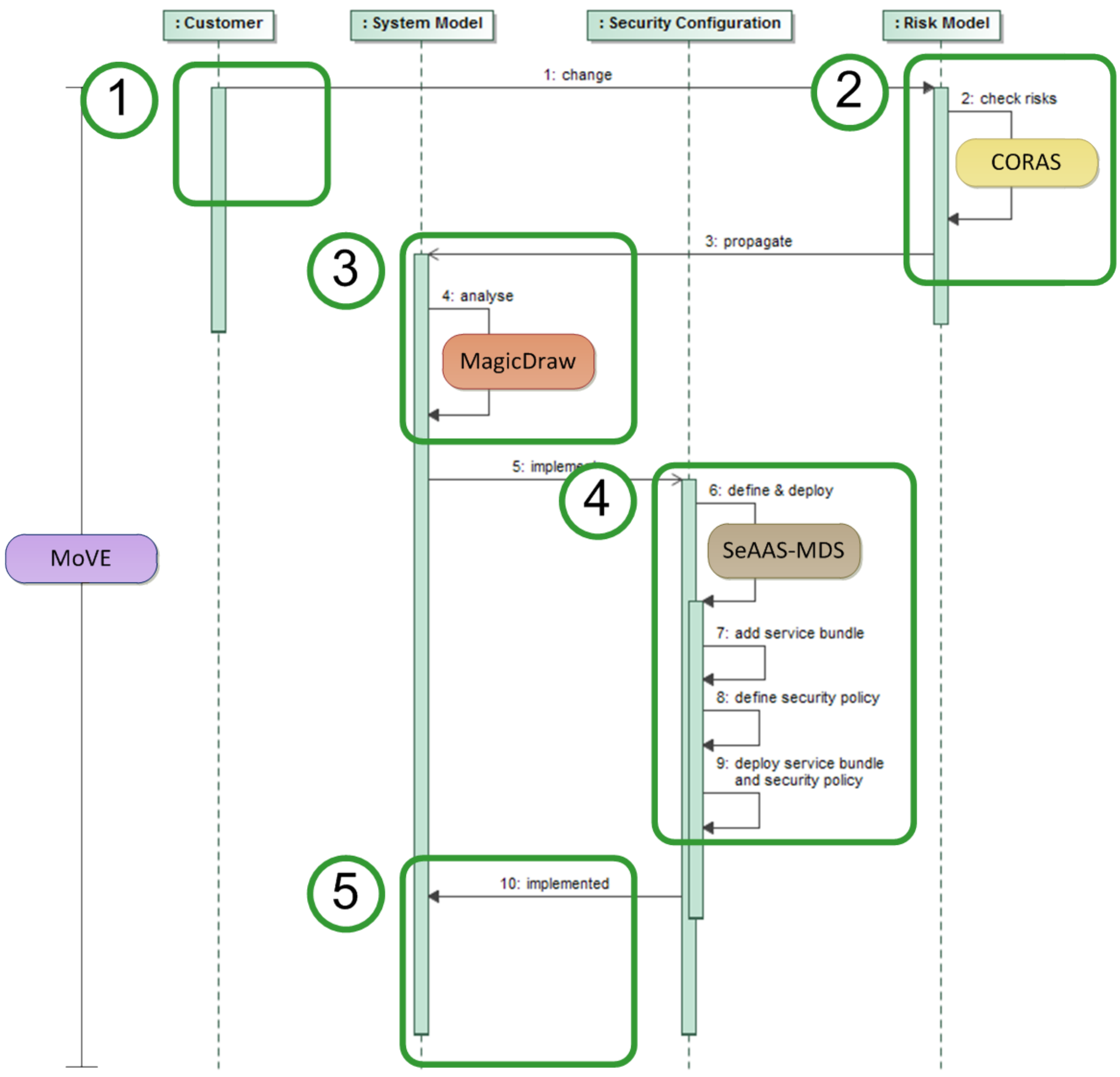


Figure 2 Integrated Process change story for the HOMES case

The change scenario of the HOMES case study is based on an initial change event that is driven by the customer, in this specific case the HOMES gateway operator. Numerous complaints by the consumers and by third-party service providers pose a threat to the business model of the operator. Therefore a risk analysis is ordered to understand weaknesses and vulnerabilities in the current system.

Figure 2 outlines the sample change story as an instance of an Integrated Process. Please note, that the change story was simplified and does not include the Test Model that was part of the initial version of the change story outlined in D.2.2 (cf. [31], Figure 91). The security configuration lane has been added to outline how the model-driven configuration interface of the Security-as-a-Service-Architecture (SeAAS-MDS) can be integrated in the overall change handling process.

After the ordering of a new risk analysis the risk management team updates the risk model (step 2) to analyze and understand the scenarios leading to the consumer and third-party complaints. As a treatment to manage the identified risks and underlying vulnerabilities the deployment of a non-repudiation protocol is proposed. After the conclusion of this task the risk model is committed to the MoVE tool which in the background updates the states according to the defined state machines. In addition the MoVE tool triggers the next action, which is the system analysis.

In step 3 the system designer analyzes the current model of the HOMES gateway system and changes the system model to reflect the implementation of the new treatment “deployment of non-repudiation protocol”. After the update of the system model he then orders the implementation of the non-repudiation protocol to the security configuration team. Again, the system modeler commits the new version of the system model to the MoVE tool which updates the state of the model and triggers the next action, which in this step is the security configuration.

The security configuration is handled with the model-driven security configuration interface for the Security-as-a-Service-Architecture (SeAAS-MDS) which supports the definition of policies and protocols (step 4). The defined protocol is then transformed to configuration files which are deployed in the Security-as-a-Service Engine (SeAAS) integrated in the HOMES gateway. After the successful definition of new security policies the security configuration is committed to the MoVE tool which updates the state. In addition the MoVE tool after the successful implementation of the additional security service bundle and the respective security policy sends a notification to the system modeler (step 5).

Figure 3 outlines the different tools that are connected in this scenario using MoVE in the backend. We use a UML system modeling tool (i.e. MagicDraw), a spreadsheet representation of CORAS risk models, and a model-driven configuration interface for the Security-as-as-Service-Architecture (SeAAS-MDS). We choose MagicDraw as an UML diagramming tool because we use it heavily in our research group. The decision for using a spreadsheet representation of the CORAS risk models was based on the estimation of the effort required to realize and implement an adaptor integrated directly in CORAS. In principle we could have used also alternative tools and models.

In the following the overall change story connecting these three tools and the respective user or stakeholders are outlined. The Integrated SecureChange Process as a light-weight variant of a change-driven security engineering process supports only a coarse-grained coupling of the different models that are processed with the three tools. A fine-grained coupling based on the states of single model elements was realized and demonstrated for the second ATM validation workshop.

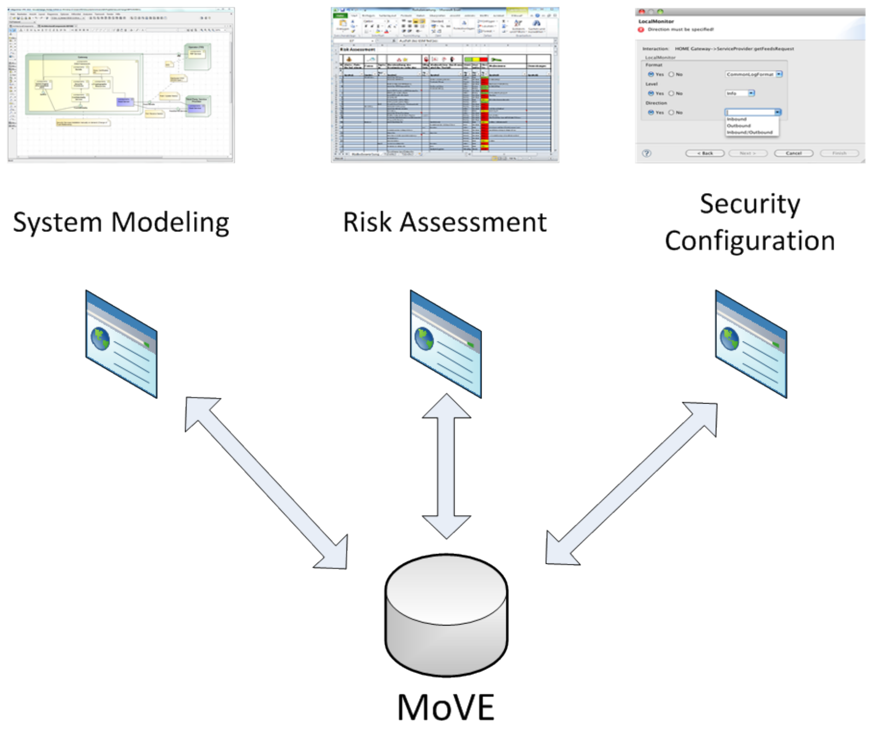


Figure 3 Overview of the tools used in the HOMES case

Figure 4 reflects the overall architecture and outlines which tools have been connected with the MoVE tool in the handling of the HOMES case study.

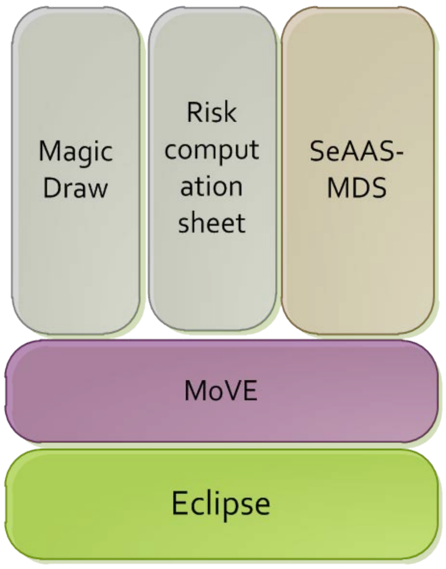


Figure 4 Tools integrated in change handling of the HOMES case

2.3.1 Overview of system and risk models before the change

The system model before change is depicted in Figure 5. At this point we have a running HOMES gateway connecting the operator’s services, third-party service providers and the consumers using the HOMES gateway. The HOMES gateway is already equipped with a Security-as-a-Service Engine.

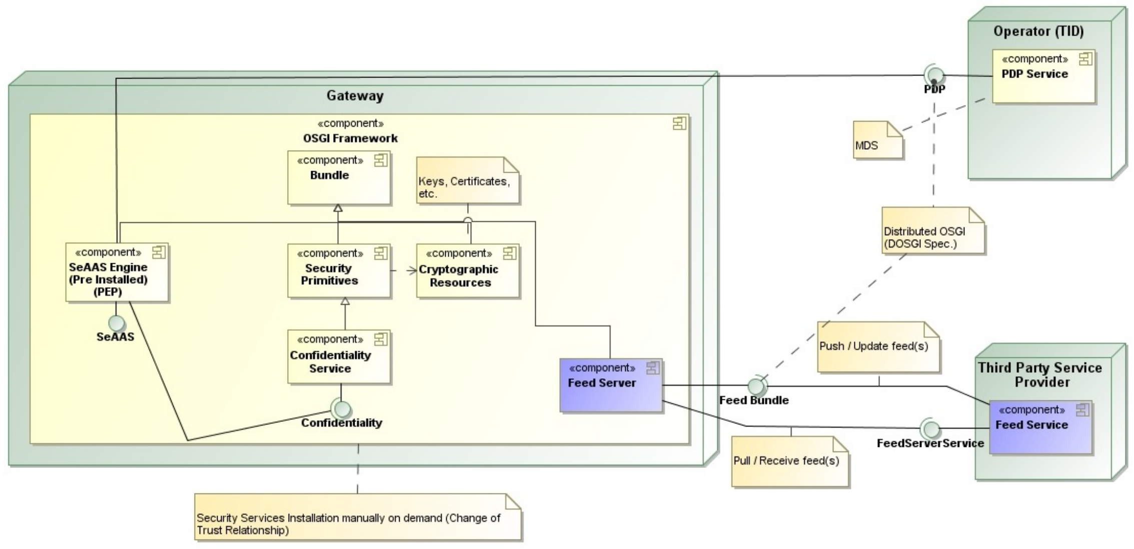

Operator (TID)
«component» PDP Service
PDP
MDS
Distributed OSGI (DOSGI Spec.)
Gateway
«component» OSGI Framework
«component» Bundle
Keys, Certificates, etc.
«component» SeAAS Engine (Pre Installed) (PEP)
SeAAS
«component» Security Primitives
«component» Cryptographic Resources
«component» Confidentiality Service
Confidentiality
«component» Feed Server
Feed Bundle
Push / Update feed(s)
Third Party Service Provider
«component» Feed Service
FeedServerService
Pull / Receive feed(s)
Security Services Installation manually on demand (Change of Trust Relationship)

Figure 5 UML system diagram before change

The initial version of the CORAS risk models before any change handling is depicted in Figure 6. While in principle it is possible to write a specific adaptor for the CORAS tool to directly process CORAS models with the MoVE tool, we have used a spreadsheet representation of the same models. That way we can use a generic CSV file for representing and processing the models.

Table 1 contains the same risk models as outlined in Figure 6 CORAS risk diagram before change, but represented as a spreadsheet. To handle the overall change process, we have developed an adaptor for the MoVE tool which is able to handle the risk model in the format of a spreadsheet.

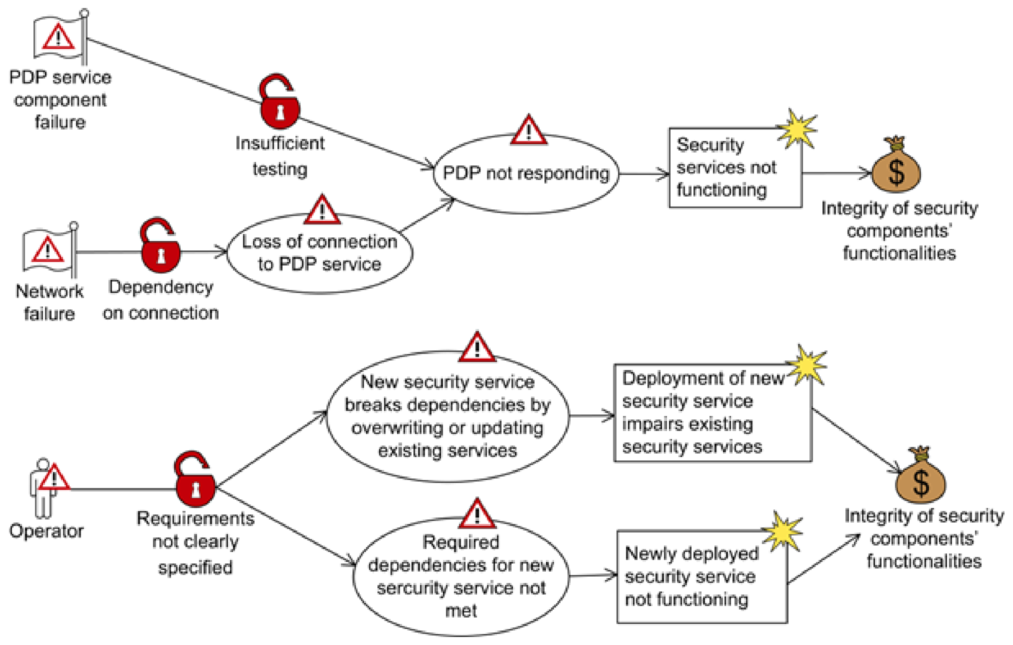

PDP service component failure
Insufficient testing
PDP not responding
Security services not functioning
Integrity of security components' functionalities
Network failure
Dependency on connection
Loss of connection to PDP service
New security service breaks dependencies by overwriting or updating existing services
Deployment of new security service impairs existing security services
Operator
Requirements not clearly specified
Integrity of security components' functionalities
Required dependencies for new sercurity service not met
Newly deployed security service not functioning

Figure 6 CORAS risk diagram before change

Table 1 Spreadsheet representation of CORAS risk model before change

Asset	**ID**	**Unwanted incident/threat scenario**	**Leads to**	**Vulnerability**	**Threat agent**	**Treatment**
Integrity of security components functionalities	UI1	Security Services not functioning				
	TH1	PDP not responding	UI1	Insuffcient testing	PDP service component failure	
	TH2	Loss of connection to PDP service	TH1	Dependency on connection	Network failure	
	UI2	Deployment of new security service impairs existing security services				
	TH3	New security service breaks dependencies by overwriting or updating existing serivces	UI2	Requirements not clearly specified	Operator	
	UI3	Newly deployed security service not functioning				
	TH4	Required dependencies for new security service not met	UI3	Requirements not clearly specified	Operator	

2.3.2 Step 1: Customer orders a new risk analysis

The customer, which in this case is the operator of the HOMES gateway, orders a new risk analysis after increasing complaints from consumers and third-party service providers. The risk analysis team checks out the latest risk model from the MoVE repository. The MoVE tool changes the state of the risk model from “complete” to “defined” (cf. Figure 7). As soon as the risk analysis team starts updating the risk model, the state changes again to “checking risks”.

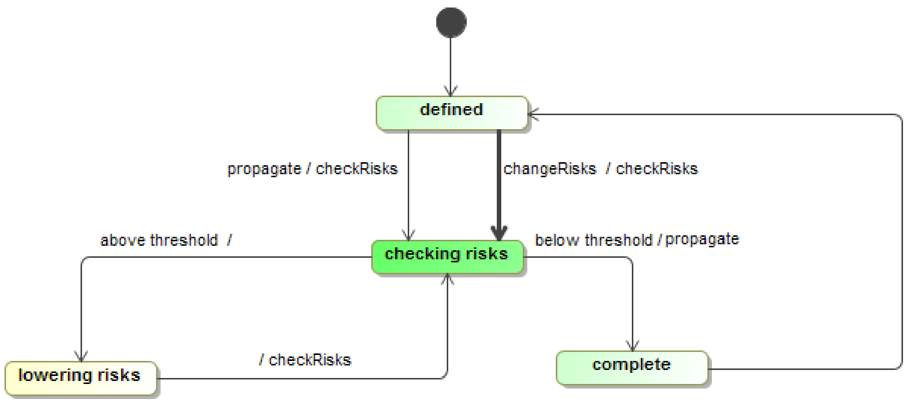
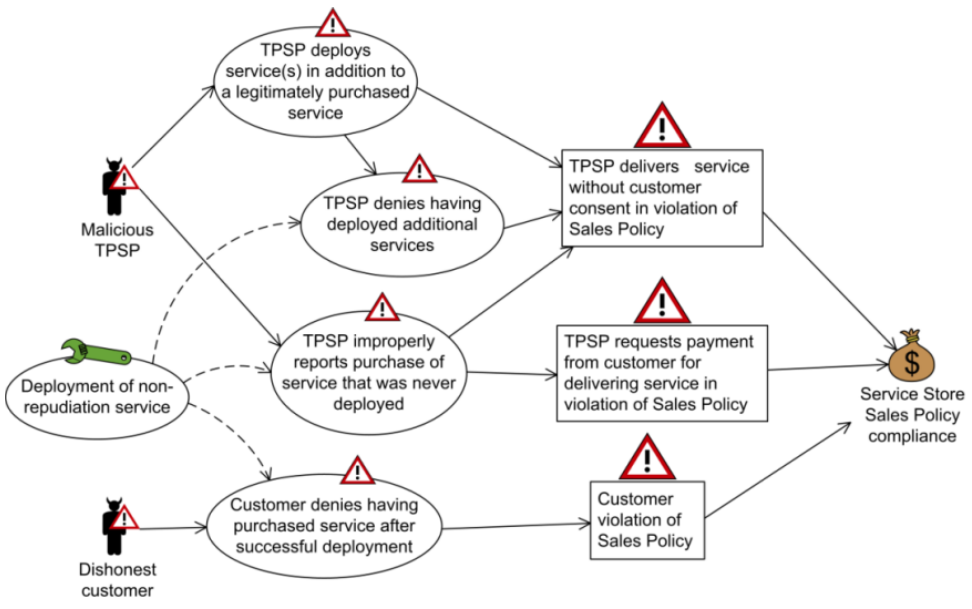


Figure 7 The state machine reflecting the life cycle of the risk model

2.3.3 Step 2: Update of the risk model

The result of the updated risk analysis is an extension of the existing risk model with new scenarios that threaten the service store sales policy compliance (cf. Figure 8). In Figure 8 the treatment for addressing three threat scenarios is also depicted, namely the deployment of a non-repudiation service in the HOMES gateway. Table 2 represents the risk model outlined in Figure 8 in a spreadsheet.



TPSP deploys service(s) in addition to a legitimately purchased service
TPSP delivers service without customer consent in violation of Sales Policy
Malicious TPSP
TPSP denies having deployed additional services
TPSP improperly reports purchase of service that was never deployed
TPSP requests payment from customer for delivering service in violation of Sales Policy
Deployment of non-repudiation service
Service Store Sales Policy compliance
Customer denies having purchased service after successful deployment
Customer violation of Sales Policy
Dishonest customer

Figure 8 CORAS risk diagram addition after change

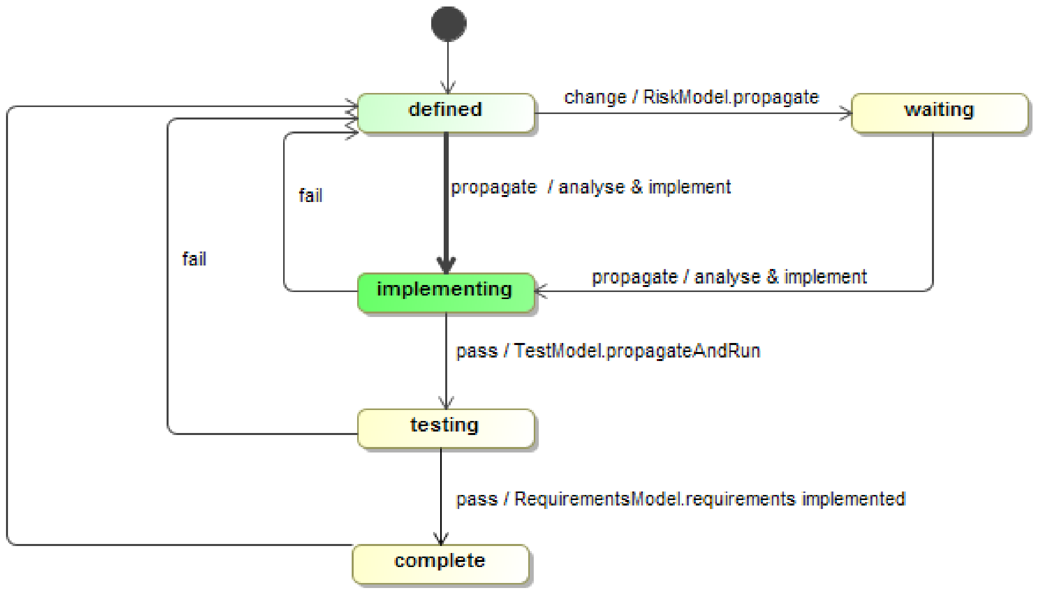
After the update of the risk model and assuming the resulting risks are below the threshold, the risk analysis team commits the updated risk model to the MoVE tool. MoVE then switches its state to “complete” (cf. Figure 7).

Table 2 Spreadsheet representation of CORAS risk model after change

Asset	**ID**	**Unwanted incident/threat scenario**	**Leads to**	**Vulnerability**	**Threat agent**	**Treatment**	**Cures**
Service Store Sales Policy compliance	UI4	TPSP delivers service without customer consent in violation of Sales Policy					
Service Store Sales Policy compliance	TH5	TPSP deploys service(s) in addition to a legitimately purchased service	UI4, TH6		Malicious TPSP		
Service Store Sales Policy compliance	TH6	TPSP denies having deployed additional services	UI4, TH6			Deployment of non-repudiation	TH6
Service Store Sales Policy compliance	UI5	TPSP requests payment from customer for delivering service in violation of Sales Policy					
Service Store Sales Policy compliance	TH7	TPSP improperly reports purchase of service that was never deployed	UI4, UI5		Malicious TPSP	Deployment of non-repudiation	TH7
Service Store Sales Policy compliance	UI6	Customer violation of Sales Policy					
Service Store Sales Policy compliance	TH8	Customer denies having purchased service after successful deployment	UI6		Dishonest customer	Deployment of non-repudiation	TH8

2.3.4 Step 3: System analysis of the change

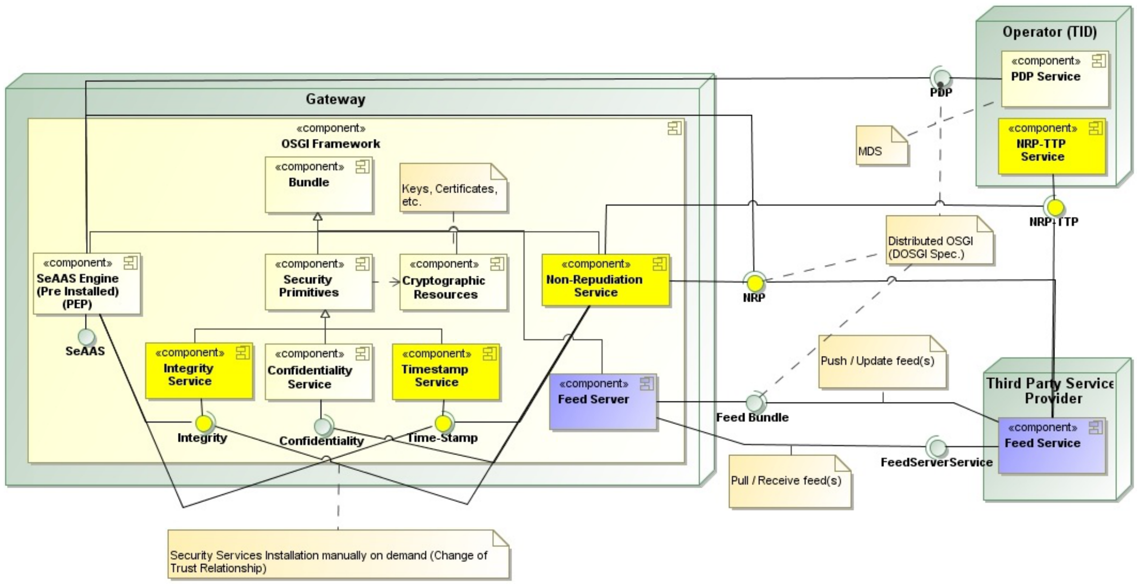
After the commit of the updated risk model in step 2 the MoVE tool triggers the next action which is the system analysis. The system analyst can use his UML system modeling tool to analyze the current architecture of the HOMES gateway before ordering the implementation of the non-repudiation protocol. The first step he does is to update his current system model from the MoVE repository. The state of this updated system model was switched to “defined” by MoVE during the change propagation.



defined
change / RiskModel.propagate
waiting
fail
propagate / analyse & implement
fail
implementing
propagate / analyse & implement
pass / TestModel.propagateAndRun
testing
pass / RequirementsModel.requirements implemented
complete

Figure 9 The state machine reflecting the life cycle of the system model

Figure 5 showed the architecture of the HOMES gateway before the implementation of the proposed treatment “deployment of non-repudiation protocol”. The HOMES gateway is equipped with a Security-as-a-Service-Engine (SeAAS) and has a few security services already deployed. At this point the system analyst updates the system model to reflect the changes introduced due to the planned deployment of a non-repudiation protocol (cf. Figure 10). One can see that there are additional security services to be deployed on the HOMES gateway and at the operator site.



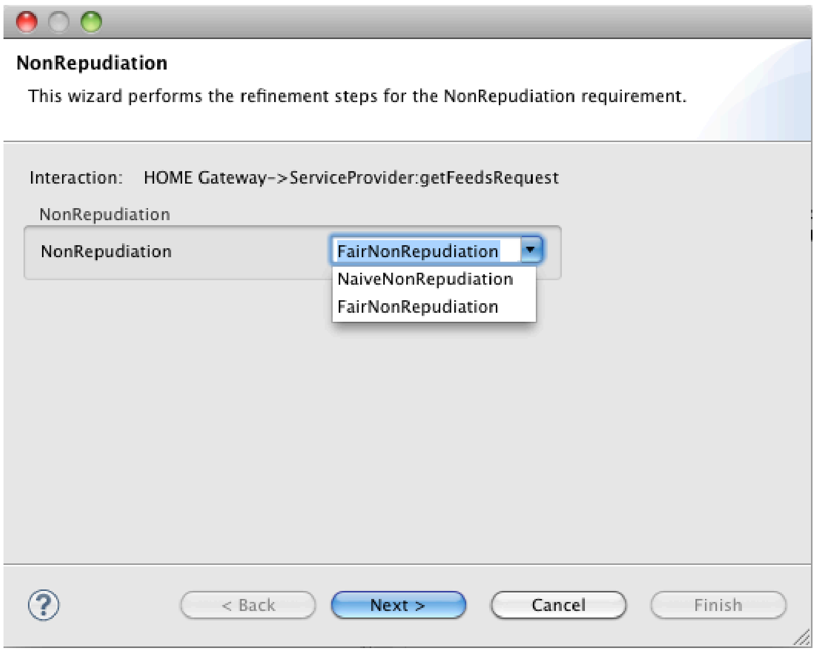
Gateway
«component» OSGI Framework
«component» Bundle
Keys, Certificates, etc.
«component» SeAAS Engine (Pre Installed) (PEP)
SeAAS
«component» Security Primitives
«component» Cryptographic Resources
«component» Non-Repudiation Service
«component» Integrity Service
«component» Confidentiality Service
«component» Timestamp Service
«component» Feed Server
Integrity
Confidentiality
Time-Stamp
Operator (TID)
«component» PDP Service
«component» NRP-TTP Service
PDP
MDS
NRP-TTP
NRP
Distributed OSGI (DOSGI Spec.)
Push / Update feed(s)
Feed Bundle
Third Party Service Provider
«component» Feed Service
FeedServerService
Pull / Receive feed(s)
Security Services Installation manually on demand (Change of Trust Relationship)

Figure 10 UML system model after change

After the system analyst concludes his analysis and confirms the changes, he will commit the new version of the system model again to MoVE. MoVE changes the state of the system model to “implementing” and triggers an activity on the side of the security configuration team (cf. Figure 9).

2.3.5 Step 4: Change of security configuration and deployment

After the commit of the new system model by the system analyst the MoVE tool hands over control to the security configuration team. The state of the security configuration model is changed to “defined” by MoVE. The security configuration team is actually responsible for the implementation and realization of the non-repudiation protocol. The first step carried out is to update their current local security configuration with the new version from the MoVE repository.


NonRepudiation

This wizard performs the refinement steps for the NonRepudiation requirement.

Interaction: HOME Gateway->ServiceProvider:getFeedsRequest

NonRepudiation

NonRepudiation: FairNonRepudiation

Figure 11 Change to the configuration of security services

The security configuration is changed using the model driven security configuration interface of the SeAAS (cf. Figure 11). In the case that the required security service is not available yet, the security configuration team can extend the SeAAS with a new security protocol. The extensibility of SeAAS is described in more detail in Section 4.2.2. After the security configuration team concludes the reconfiguration task with the SeAAS-MDS, it commits the new security configuration to the MoVE tool. MoVE then changes the state of the security configuration to “complete” and triggers the next activity.

2.3.6 Step 5: Notifying the system analyst of the changed security configuration

At step 5 the system analyst receives a notification by the MoVE tool of the completed changes of the security configuration. He updates his current risk model with an updated version from the MoVE repository. The system model has changed its state to “completed” since the new security services are now configured and deployed. In principle now the MoVE tool could trigger additional actions. For example the testing team could be notified to start a test run on the updated configuration. Another example for an additional change handled by MoVE could be an update to the risk model to reflect the risk reduction realized by the implemented treatment. To keep the application of MoVE on the HOMES case study simple at this point the overall change handling process is concluded.

This handling of the overall HOMES change story reflects a coarse-grained change driven engineering process. Such a coarse-grained process allows orchestrating proprietary tools and processes as changes are reflected on the basis of entire models. On the ATM case study a fine-grained change driven engineering process was applied, which is based on the states and changes of single model elements. Both approaches are supported by the MoVE tool. The evaluation feedback of the MoVE tool is outlined in Section 5.

3 MoVE Tool

In this section we describe the MoVE Tool. MoVE, the shortcut of Model Versioning and Evolution, is a tailor-made model repository to support Living Models and change-driven process. In the following subsections we describe MoVE's objectives and features in Section 3.1, the architecture of MoVE in Section 3.2 and its extendibility and configuration in Section 3.3.

3.1 Objectives and Features

The main objective of MoVE is to support a model based, change driven process to bring the concept of Living Models into being. In order to fulfill this objective MoVE implements several key features. In the following the most important ones are listed and shortly explained. For more details about the concepts behind, please see D2.2 (cf. [31]).

Model Versioning. To be able to support efficiently model based development, we need a sophisticated model versioning. Our repository, based on SVN, enables the user to store models and recap arbitrary versions of models, just as it is possible for other files in SVN. It is also possible to concurrently edit one version of a model. By committing any version to the repository, an implemented model merging mechanism gets active and merges the models, if needed. If there are some conflicts that cannot be handled automatically, the modeler gets informed by MoVE.

Stakeholder Views and Collaboration, as well as Tool Support. In a project of a certain size several stakeholders with different roles are involved. There might be security engineers as well as system designers and software architects. It is obvious that they do not use the same tools and do not have the same view on the system, as they are interested in different aspects. MoVE supports the usage of different tools and provides also different views for stakeholders. Since all artifacts are stored in the very same repository, collaboration is easily possible. This collaboration is also enhanced by the next two implemented features.

Change Driven Engineering. Change driven engineering combines three aspects: *states*, *change propagation* and *support of state machines*. Every element stored in the repository has a state-attribute. And every class of elements has an own state machine. If a state of an element changes, the according state machine gets active and looks if some transitions can fire. As long as there are state changes possible, they are performed in a loop. The termination conditions of the loop are to reach stable states of all elements. Through the enactment of one state machine, several state machines of other elements can be triggered. This is called change propagation, as the change is propagated from one model element to others.

Partial Model Support. The overall system model is the union of several partial models, e.g. data base model, security model, etc. We need the concept of partial models to be able to efficiently support various stakeholders. By introducing partial models, MoVE has to cope with a heterogeneous environment. Each partial model has a *master tool*, which is the tool that was used for its creation. For example, in our context a UML model is created with the tool Magic Draw, while a security model is defined with the help of Microsoft Excel. For each used tool, MoVE has to provide an adapter to be able to work with various formats. The intersection of all partial models may not be empty. The conflict resolution on overlapping task is done by MoVE.

3.2 Architecture

The architecture of MoVE was described in Deliverable 2.2 (cf. [31]). We therefore summarize here the most important concepts of MoVE's architecture.

MoVE is a component based model repository consisting of a basic component for storing EMF models leveraging on SVN to provide features of standard version control systems. To provide an interface for easy integration of MoVE into modeling software such as MagicDraw (cf. [15]), a client-side component called *MoVEClient* hides all protocol calls between client and server. The *MoVEClient* also offers a rich API, useful for integration into modeling tools, having features such as model merging or update notification.

The main component of MoVE on the server-side is called *MoVEServer*. From the technological point of view the *MoVEServer* component is an Eclipse RCP application that provides extension points for plugins following the Eclipse plugin design principles. The underlying SVN server informs the *MoVEServer* on every change triggered by the *MoVEClient*. Starting with this notification the *MoVEServer* analyses each change of a model and propagates this information to plugins that are associated to the respective model and the type of change that was made.

In Figure 12 you can see the client-server architecture of MoVE. On the server-side several plugins use the *PluginInterface* interface to add new functionalities to MoVE. One example is the state machine plugin (*StatemachinePlugin*) that allows MoVE to execute and store state machines for certain elements. Deliverable 2.2 (cf. [31]) provides a closer look on this plugin. In the next section we show possibilities to extend MoVE.

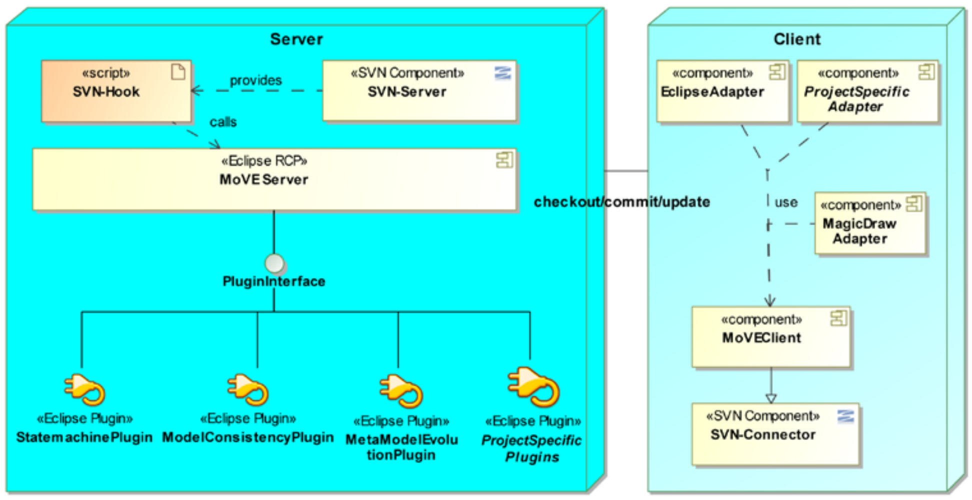


Figure 12 The architecture of MoVE

3.3 Extendibility and configuration

MoVE is designed as a very flexible and extensible framework which is highly configurable. As described in Section 3.2, MoVE has client- and server-side components; both sides can be extended using the provided APIs and interfaces.

To show the feasibility and efforts to create MoVE extensions the following Sections will demonstrate how to write three different extensions of MoVE.

- Section 3.3.1 explains how to write a client side adapter that integrates MoVE into the CORAS tool using the *MoVEClient* API.
- Section3.3.2 shows two examples of server-side extensions. The first extension adds a new state machine to a project. The second extension adds a small metric plugin that uses OCL to count the number of *Information* instances attached to a model element referencing the models presented in Deliverable 2.2 (cf. [31]).

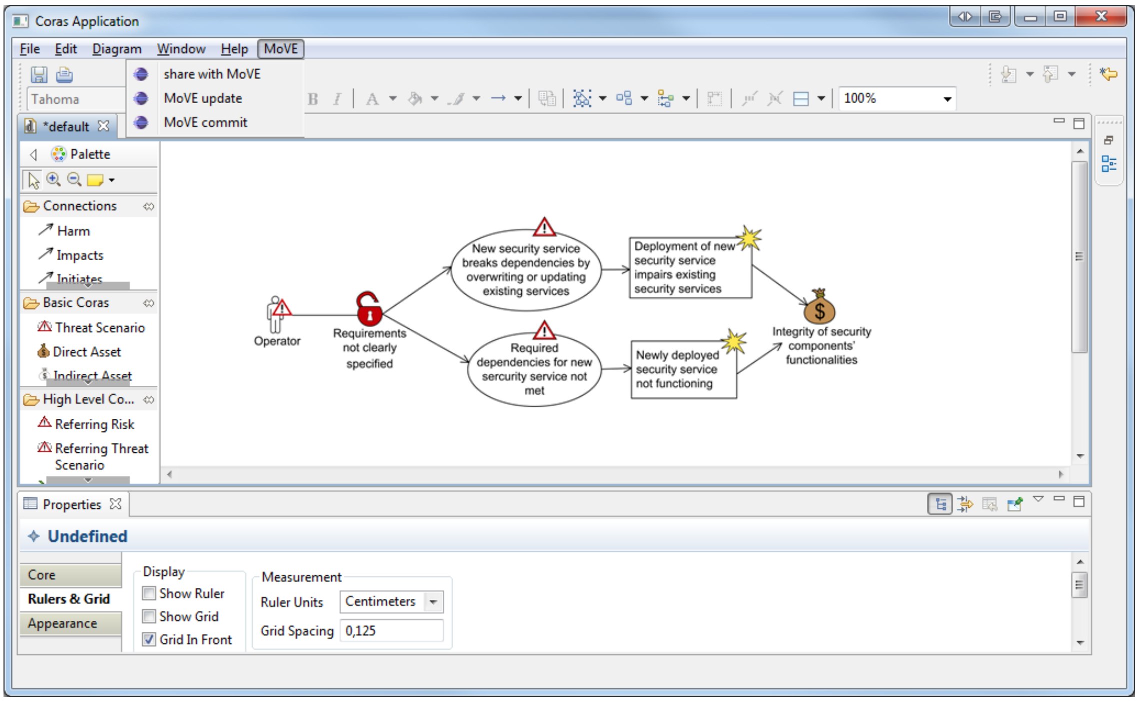
The section is finished with an explanation of MoVE's Administrator Interface that allows adding new plugins and configuring them for a certain type of model and change-event.

3.3.1 Client-side extension

To integrate modeling tools into MoVE a developer must implement a MoVE adapter. Adapters are usually plugins integrated into modeling tools using the API provided by the tools. The functionality to communicate with MoVE is provided by the *MoVEClient* component. For example the Magic Draw adapter uses the Magic Draw Open API to create Menus and Windows in the Magic Draw Environment but it uses the *MoVEClient* to commit models to the MoVE repository. The effort of creating an adapter strongly depends on the API provided by the tools. The main challenge of an adapter is to convert the data into an XMI file using the (EMF) UML metamodel.

Modeling tools in the Eclipse environment, such as Papyrus, are usually based on EMF and therefore use models which do not need any conversion. MoVE provides an adapter for Eclipse which can be integrated into any Eclipse application.

Figure 13 shows a mock-up screenshot on how the integration of MoVE into the CORAS tool could look like. This screenshot is used to show the conceptual integration of MoVE and CORAS. The MoVE Eclipse adapter already comes with a graphical user interface such as the menu depicted in the mock-up screenshot. A developer of a new adapter for modeling software can use this adapter as an abstract component to build the concrete adapter upon it. The developer needs to decide what data the adapter should sent to MoVE and how the results of MoVE influence the current model. For example a developer needs to implement a method that changes the color of a state variable in case MoVE altered the state. This is adapter specific code which must be implemented for each tool.


Coras Application
File Edit Diagram Window Help MoVE
share with MoVE
MoVE update
MoVE commit
Tahoma
*default
100%
Palette
Connections
Harm
Impacts
Initiates
Basic Coras
Threat Scenario
Direct Asset
Indirect Asset
High Level Co...
Referring Risk
Referring Threat Scenario
Operator
Requirements not clearly specified
New security service breaks dependencies by overwriting or updating existing services
Required dependencies for new sercurity service not met
Deployment of new security service impairs existing security services
Newly deployed security service not functioning
Integrity of security components' functionalities
Properties
Undefined
Core
Rulers & Grid
Appearance

Display
- [ ] Show Ruler
- [ ] Show Grid
- [x] Grid In Front

Measurement

Ruler Units	Centimeters
Grid Spacing	0,125

Figure 13 A screenshot of the MoVE plugin integrated in the CORAS tool

3.3.2 Server-side extension

3.3.2.1 Add new state machine

Adding a new state machine enhances the change driven behavior of MoVE. If we want to create a new state machine we need an Eclipse instance containing the *MoVE Admin View*, available as an Eclipse plugin. A screen shot of this Eclipse perspective is depicted in Figure 14.

On the left side of this figure one can see the project explorer with the imported modeling projects, also containing a folder named "*Statemachines*". Within this folder all existing state machines are defined. We will now add a new state machine by creating another SCXML file in this folder. We have to take care that the state machine file is named according to the related model element in the meta model. E.g. adding a state machine for the model element "Information", we have to name the according SCXML "information.scxml".

As soon as the file is created, we can define the desired behavior of the state machine by using SCXML and OCL. After editing the file, we commit the changes to the MoVE repository. At this point, the newly defined state machine is considered during MoVE change handling. For an example of a SCXML statemachine we refer to the Deliverable D2.2, Section 4.3.2.2, Listing 1 [31].

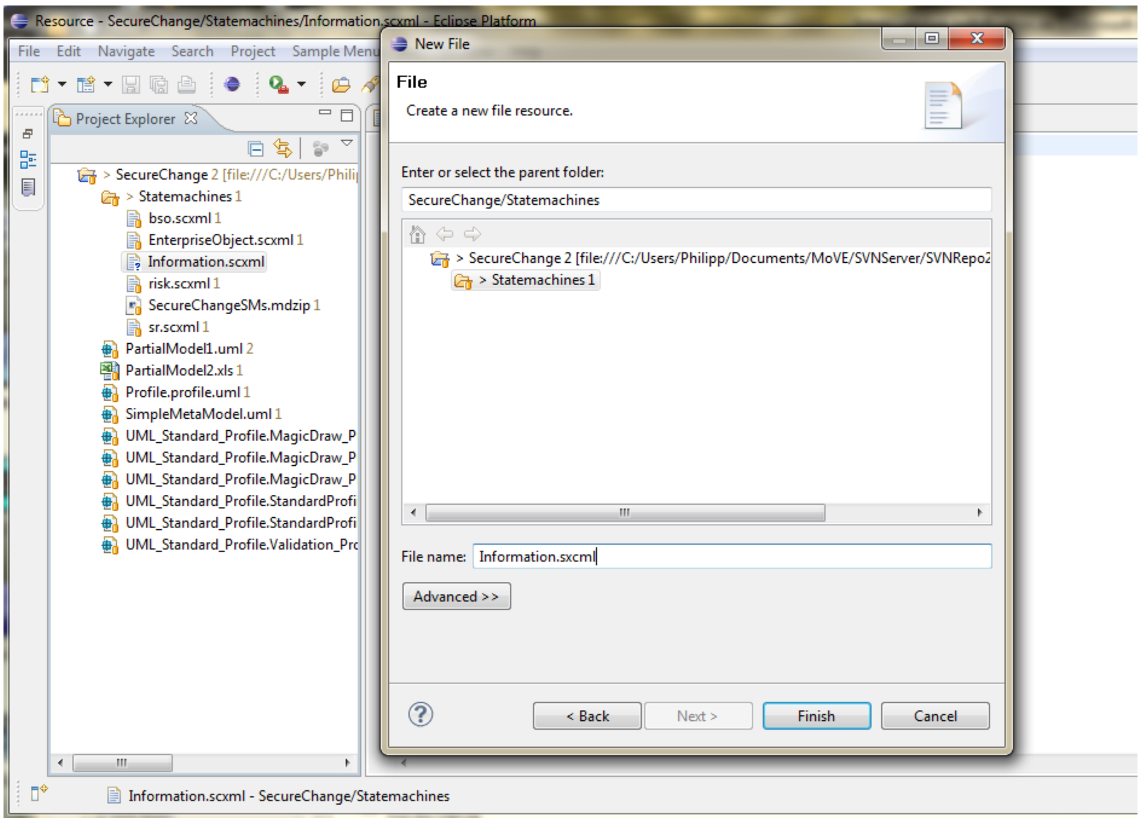

Resource - SecureChange/Statemachines/Information.scxml - Eclipse Platform
File Edit Navigate Search Project Sample Menu
Project Explorer
SecureChange 2 [file:///C:/Users/Phili
Statemachines 1
bso.scxml 1
EnterpriseObject.scxml 1
Information.scxml
risk.scxml 1
SecureChangeSMs.mdzip 1
sr.scxml 1
PartialModel1.uml 2
PartialModel2.xls 1
Profile.profile.uml 1
SimpleMetaModel.uml 1
UML_Standard_Profile.MagicDraw_P
UML_Standard_Profile.MagicDraw_P
UML_Standard_Profile.MagicDraw_P
UML_Standard_Profile.StandardProfi
UML_Standard_Profile.StandardProfi
UML_Standard_Profile.Validation_Pro
New File
File
Create a new file resource.
Enter or select the parent folder:
SecureChange/Statemachines
SecureChange 2 [file:///C:/Users/Philipp/Documents/MoVE/SVNServer/SVNRepo2
Statemachines 1
File name: Information.sxcml
Advanced >>
< Back
Next >
Finish
Cancel
Information.scxml - SecureChange/Statemachines

Figure 14 Adding a new state machine (X1)

3.3.2.2 Add a plugin to check a metric

In this section we show how to add a plugin that calculates a metric for a changed model. The result of the metric is then returned in a message delivered to the user automatically via client-side adapters. In this example the model was changed in MagicDraw and, using the *MoVEadapter* for MagicDraw, committed to MoVE. The resulting message is shown in MagicDraw after the commit was done.

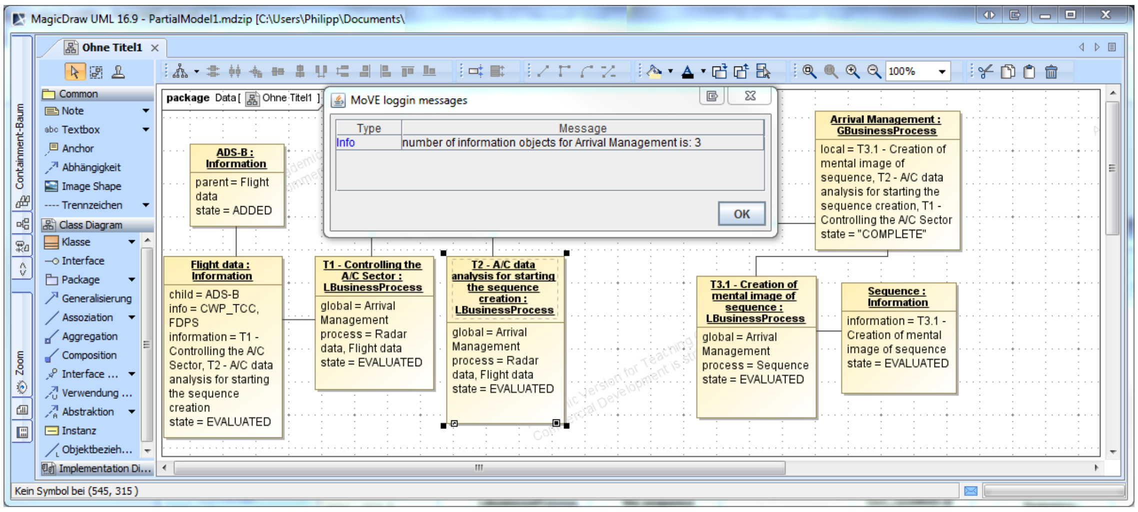
The basic idea of MoVEplugins was already described in Deliverable 2.2 (cf. [31]) therefore we will only shortly summarize the concepts. Every time a model is committed to the MoVE repository, the MoVE server analyses the changes of the model between the committed version and the previous version. For each change MoVE generates a change event that contains information about the type of model element that was changed and the type of change itself. The events are sent to the plugin registry which is a component that allows MoVEplugins to register for a certain type of change event. An example for a change event is *MoVEConstants.EVENT_TYPES.PRE_COMMIT* which is triggered every time a model was changed irrespective of the type of the change. The plugin registry can be configured with a graphical user interface which is described in the end of this section.

```
package at.qe.move.ATMMetric;
import …;
import at.qe.move.svnbackendrcp.plugin.MoVEPluginInterface;
public class MetricPlugin extends MoVEPluginInterface {
public static String VERSION_ID = "1.0.0";
private OCLHelper<Classifier, ?, ?, Constraint> oclHelper;
private OCL<?, Classifier, ?, ?, ?, ?, ?, ?, ?, Constraint, Class, ?> ocl; // generics
//Returns all information objects starting from a Global Business Process
private String treeIterator_GProcess_Info = "package Data context
model::GBusinessProcess def treeIterator_GProcess_Info : treeIterator_GProcess_Info() :
Set(model::Information) = Set{}->union(self.local->iterate(obj :
model::LBusinessProcess; result : Set(model::Information) = Set{} | result-
>union(obj.process))) endpackage" ;
@Override
public void handleNotification(String notification, Object payload) {
ModelVersion version = (ModelVersion) payload;
if (version.getAdapter() != null)
{
EObject model = version.getAdapter().getModel();
model.eResource().getResourceSet().getPackageRegistry().put(UMLPackage.eNS_URI,
UMLPackage.eINSTANCE) ;
ocl = OCL.newInstance(new
UMLEnvironmentFactory(model.eResource().getResourceSet().getPackageRegistry(), model
         .eResource().getResourceSet()));
oclHelper = ocl.createOCLHelper();
try {
ocl.parse(new OCLInput(treeIterator_GProcess_Info)) ;
} catch (ParserException e1) {
Activator.LogError("problem parsing query "  );
return;
}
//check all GBusinessProcess objects
TreeIterator<EObject> iterator = model.eAllContents();
while (iterator.hasNext())
{
EObject item = iterator.next();
if (item.eClass().getName().equals("GBusinessProcess))"))
{
EObject oclContext = getContextForObject("GBusinessProcess");
oclHelper.setInstanceContext(oclContext);
Object result = null;
try {
OCLExpression<Classifier> query = null;
query = oclHelper.createQuery("result: Integer = treeIterator_GProcess_Info()->size");
result =  ocl.evaluate(item, query);
//return result as information
Activator.LogInfo("number of information objects for " + ((NamedElement)item).getName()
+
     " is: " + result);
} catch (ParserException e) {
Activator.LogError("problem parsing query "  );
}}}
}
}
@Override
public String getName() {
return "ATM Metric Plugin";
}
@Override
public String getPluginVersion() {
return VERSION_ID;
}
@Override
public String getEvent() {
return MoVEConstants.EVENT_TYPES.PRE_COMMIT;
}}
```

Listing 1 ATMMetric as an example for a MoVE plugin

For the demonstration of this mechanism Listing 1 shows the java code which is necessary to complete the task of writing a small metric plugin (except some missing imports from the standard java libraries). The *MetricPlugin* class extends the *MoVEPluginInterface* class and implements its abstract methods. The most important function is *handleNotification(String notification, Object payload)*. The first argument provides the type of change event which caused MoVE to execute the plugin. The second argument payload delivers an object which contains information (depending on the event). For plugins listening to the pre-commit-event this payload is a *ModelVersion* object that stores the current model, a link to its last version, its metamodel and a link to the last version of the metamodel. In the plugin a loop looks for objects of the type *GBusinessProcess* (lines number 26 to 42) and executes an OCL statement on each of these objects (lines number 37 and 38). The result of the OCL query is a number that counts information objects associated with the *GBusinessProcess* object. Using the *LogInfo* function (line number 40) the result is then sent back to MoVE as an information message.

Figure 15 shows a screenshot from the MagicDraw modeling tool that displays the result of the metric plugin to the user. The *Information* object **ADS-B** was added to the model (cf. [31]) and committed to MoVE. After the commit the result is a message dialog where the first column gives information about the type of message in this case the message is of type *Info*, which is an abbreviation for a message that only delivers general *information*. The second column shows the message itself.



MagicDraw UML 16.9 - PartialModel1.mdzip [C:\Users\Philipp\Documents\
Ohne Titel1
package Data[Ohne Titel1]
MoVE loggin messages
OK

Type	Message
Info	number of information objects for Arrival Management is: 3

ADS-B : Information
parent = Flight data
state = ADDED

Flight data : Information
child = ADS-B
info = CWP_TCC, FDPS
information = T1 - Controlling the A/C Sector, T2 - A/C data analysis for starting the sequence creation
state = EVALUATED

T1 - Controlling the A/C Sector : LBusinessProcess
global = Arrival Management
process = Radar data, Flight data
state = EVALUATED

T2 - A/C data analysis for starting the sequence creation : LBusinessProcess
global = Arrival Management
process = Radar data, Flight data
state = EVALUATED

Arrival Management : GBusinessProcess
local = T3.1 - Creation of mental image of sequence, T2 - A/C data analysis for starting the sequence creation, T1 - Controlling the A/C Sector
state = "COMPLETE"

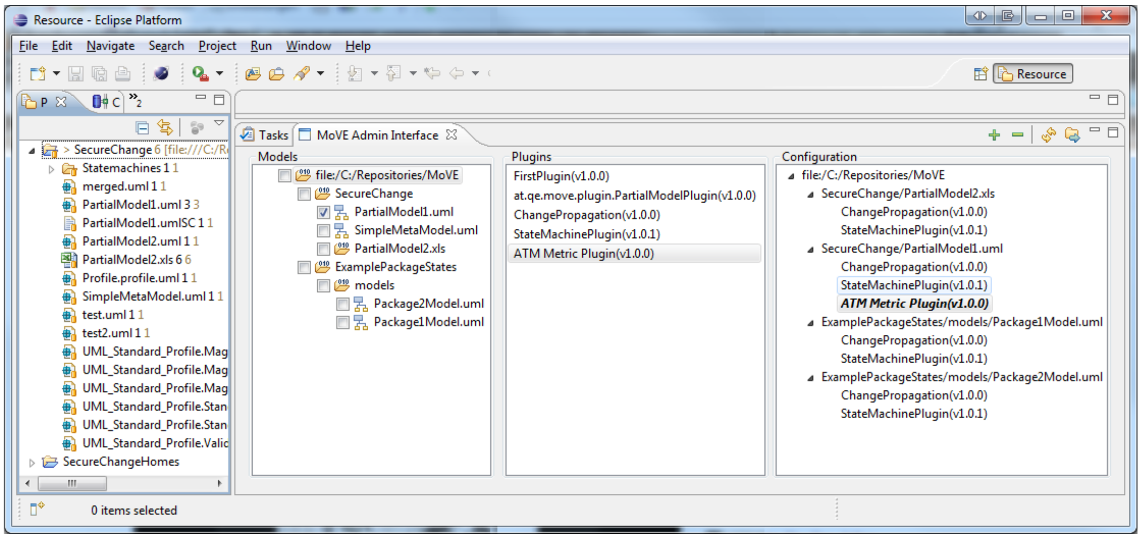
T3.1 - Creation of mental image of sequence : LBusinessProcess
global = Arrival Management
process = Sequence
state = EVALUATED

Sequence : Information
information = T3.1 - Creation of mental image of sequence
state = EVALUATED

Figure 15 Screenshot showing the result of the ATMMetric Plugin

3.3.2.3 Configuration

MoVE enables the users to easily change the configuration of models, meta models and plugins of a project. To change this configuration, the MoVE Admin Interface provides a GUI (cf. Figure 16).


Resource - Eclipse Platform

Models
file:/C:/Repositories/MoVE
SecureChange
PartialModel1.uml
SimpleMetaModel.uml
PartialModel2.xls
ExamplePackageStates
models
Package2Model.uml
Package1Model.uml

Plugins
FirstPlugin(v1.0.0)
at.qe.move.plugin.PartialModelPlugin(v1.0.0)
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
ATM Metric Plugin(v1.0.0)

Configuration
file:/C:/Repositories/MoVE
SecureChange/PartialModel2.xls
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
SecureChange/PartialModel1.uml
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
ATM Metric Plugin(v1.0.0)
ExamplePackageStates/models/Package1Model.uml
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
ExamplePackageStates/models/Package2Model.uml
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)

0 items selected

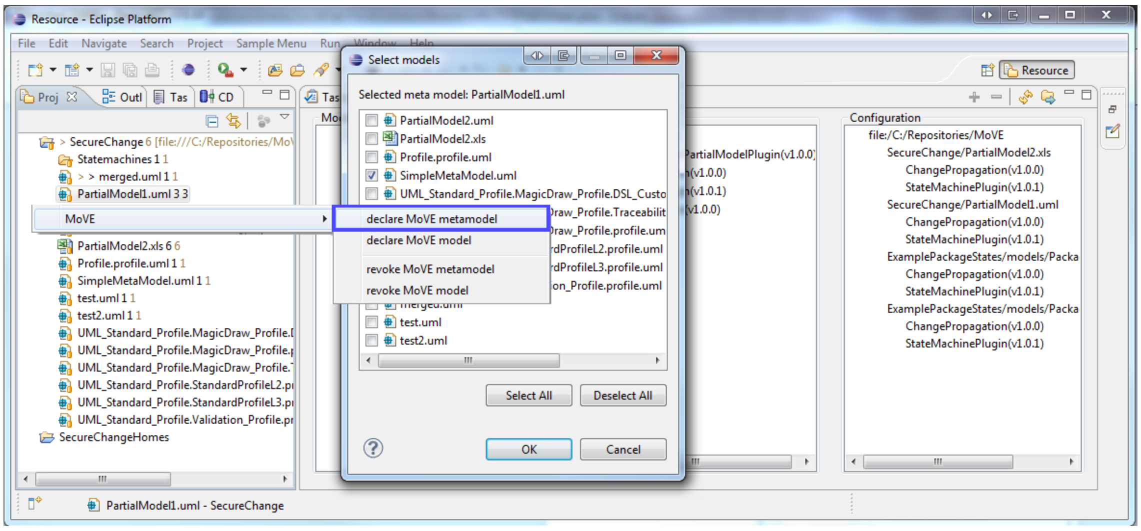
Figure 16 Screenshot of the MoVE Administration Interface

The Admin Interface includes three fields:

Models: this area is built by a file explorer, listing the content of the selected repository. One repository can contain several projects. For each project, all existing partial models are listed.

Plugins: this area lists all existing MoVE plugins for this repository.

Configuration: for each model in the repository the user can choose (configure) plugins which should be applied. In this third area of the MoVE Admin Interface the actual configuration can be viewed. To add a plugin to a model, such as the StateMachine plugin, just pull it from the middle area via drag and drop onto a model of the left area. Immediately the configuration changes. To remove a plugin from a model, select the according line in the configuration and delete it by using the context menu.


Resource - Eclipse Platform

Select models
Selected meta model: PartialModel1.uml
PartialModel2.uml
PartialModel2.xls
Profile.profile.uml
SimpleMetaModel.uml
UML_Standard_Profile.MagicDraw_Profile.DSL_Custo
test.uml
test2.uml

MoVE
declare MoVE metamodel
declare MoVE model
revoke MoVE metamodel
revoke MoVE model

Select All | Deselect All
OK | Cancel

Configuration
file:/C:/Repositories/MoVE
SecureChange/PartialModel2.xls
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
SecureChange/PartialModel1.uml
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
ExamplePackageStates/models/Packa
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)
ExamplePackageStates/models/Packa
ChangePropagation(v1.0.0)
StateMachinePlugin(v1.0.1)

PartialModel1.uml - SecureChange

Figure 17 Change configuration of MoVE metamodel

Figure 17 shows how the user can define models to be a MoVE model or choose which model is their metamodel. Declaring a model to be a MoVE model means that MoVE incorporates this model into its process. In most cases, models are marked as MoVE models by the adapter of their master tool (see Section3.1, Partial Model Support). In cases models are manually imported to MoVE projects, without the use of MoVE adapters, the models have to be declared manually as MoVE models in the following way: select a model in the project explorer (shown in Figure 17) and chose "MoVE" in the context menu. Another context menu pops up and allows the user to define the model to be a MoVE model. In the same menu the model can also be revoked to be a MoVE model.

The option "declare MoVE metamodel" opens a file selection window that allows selecting a model which should be used as the metamodel of the current model.

3.4 Comparison of MoVE with other Model Repositories

In this section we will analyze the current state of art of model repositories with similar requirements and compare them to MoVE. First of all we will start with systems that cover the versioning aspect of MoVE, followed by an analysis of current systems which support workflows.

Model versioning is the focus of several research and industry projects. One of the most popular model versioning systems is CDO[2] which has been developed as part of the Eclipse modeling framework. CDO supports model element versioning, multi user access, transactional access, scalability, thread safety and collaboration and is fully integrated into the Eclipse modeling framework (EMF). The main disadvantage of CDO is its limitation to ECore-EMF models which basically covers class diagrams solely. There is no support of instance specifications, state machines or component diagrams and no support for a change-driven process at all. Most of the current versioning systems have similar disadvantages or do not have any relation to models (for example CVS) which is crucial for model versioning systems[3].

Process-aware systems can be categorized into commercial and non-commercial tools. The most comparable non-commercial system is Unicase[4], developed by Technische Universität München. Unicase is a CASE-tool that supports modeling artifacts of a software engineering project, such as components and tasks. Its focus is on the software development process and linkage of its artifacts. Its main drawback is the fixed meta model and the limited possibility to integrate client-side tools. Integration with other (modeling) tools such as MagicDraw or the SecureChange tools is hardly possible. Unicase does not support change handling and therefore does not support a change-driven process.

[2] http://wiki.eclipse.org/CDO
[3] http://www.eternals.eu/
[4] https://teambruegge.informatik.tu-muenchen.de

IBM (Telelogic) announces Rational Doors[5] as commercial requirements management tool. Doors manages requirements in a central repository. State-based transition systems can model changes and their consequences to related requirements. The difference to our approach is that within Doors one cannot automate state changes or propagate changes automatically throughout the data. The user just gets notified which elements are affected by a specific change. Another system proposed by IBM is the Jazz-platform[6]. The Jazz platform is not a product but a platform for team-based software development based on Rational products such as IBM Rational Team Concert, Rational Quality Manager and Rational Requirements Composer. Its current strength is the linkage between artifacts using the Linked Lifecycle Data standard. Jazz does not provide any support for a change-driven process. Since the possible integration of non-Rational tools is not documented we categorize this system as a commercial system too.

A further related commercial tool is in-Step by microTOOL[7]. In-Step is a tool for process driven project management in the area of system and software development. All activities defined within the project have a status attribute. Similar to the functionality in Doors, one can trace changes of elements and also ascertain possible impacts on other elements. in-Step allows the definition of state -machines to guide the changes of each element. Change propagation and evolving the underlying meta model is not possible.

In summary the strengths of MoVE ,compared to other tools available at the moment, are the support of a generic method for model versioning, the support of tool-independent meta-models and the generic change handling mechanism.

[5] http://www-01.ibm.com/software/awdtools/doors/

[6] http://www-01.ibm.com/software/rational/jazz/

[7] http://www.microtool.de/instep/

4 Model Driven Security Interface

This section introduces an interface for the SeAAS framework which is based on the principles of Model Driven Security [29]. The goal is to provide the ability of configuring the SeAAS framework on a functional level. Therefore we implemented a prototype that is capable of generating technical policies out of functional models. So it is possible to make changes to security services on a functional level. In a last section we will present the necessary tasks to adapt our framework for Model Driven Security (MDS) to the other domains.

4.1 Framework Overview

As a basis we use workflow models describing the interaction of actors at business level. These workflow models are enhanced by security requirements. In order to be able to generate policies for the technical SeAAS platform, the initial security requirement is enhanced by further information step-by-step.

In Figure 18 the policy generation process is illustrated. The outputs of the transformation process are policies which are able to configure all participating SeAAS instances and corresponding security services.

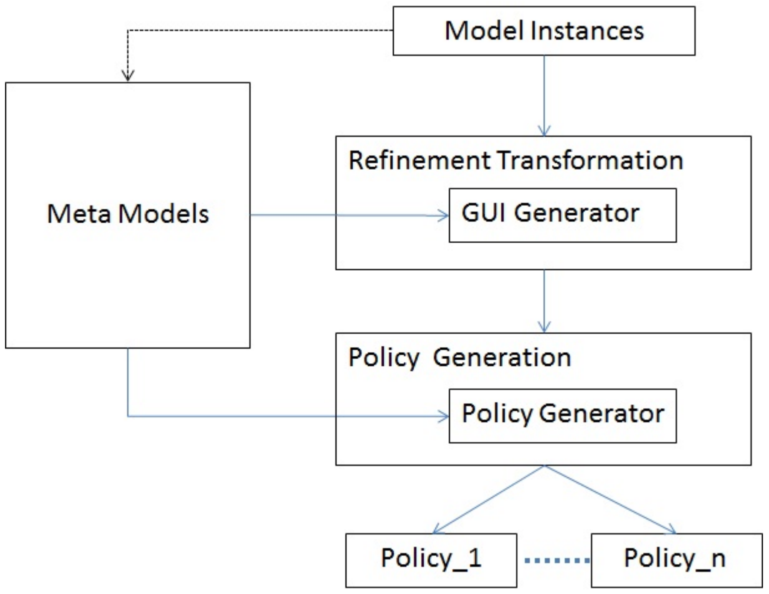


Figure 18 The policy generation process

As it is shown in Figure 18, our MDS process consists of two main steps: the refinement transformation and policy generation. Through refinement transformations, decisions concerning the type or pattern of the security service as well as architectural design options are considered. The policy generation step, on the other hand, generates security policies that configure the security services. The information about the supported security patterns and architectures are represented in the provided meta-models. These meta-models can be used to generate wizards that help performing the refinement transformation as well as the policy generation.

4.1.1 Input Models

The transformation process from Figure 18 basically needs the following three models as input:

- Workflow Model (activity diagram),
- Interface Model (class diagram),
- Document Model (class diagram).

The most important information for generating the required policy files are modeled within an UML activity diagram. This model defines both the business process between actors and the security requirements.

Our framework supports in its current version the following high level security requirements:

- integrity,
- confidentiality,
- authentication,
- non-repudiation,
- monitor,
- timestamp.

Within the activity diagram the corresponding stereotypes to these security requirements can be assigned to the data objects (messages) that are sent from one actor to another. During the transformation process they can be refined down to parameter level of a message. The authentication requirement is an exception as it can only be specified on service level.

All actors and the interactions between them are defined in an activity diagram as exemplified for HOMES in Figure 19. This example illustrates the task of retrieving feeds within our HOMES use case scenario. Here a customer behind a HOME Gateway device wants to retrieve news feeds from a Service provider and therefore the HOME Gateway sends the corresponding request (getFeedsRequest) to the Service Provider.

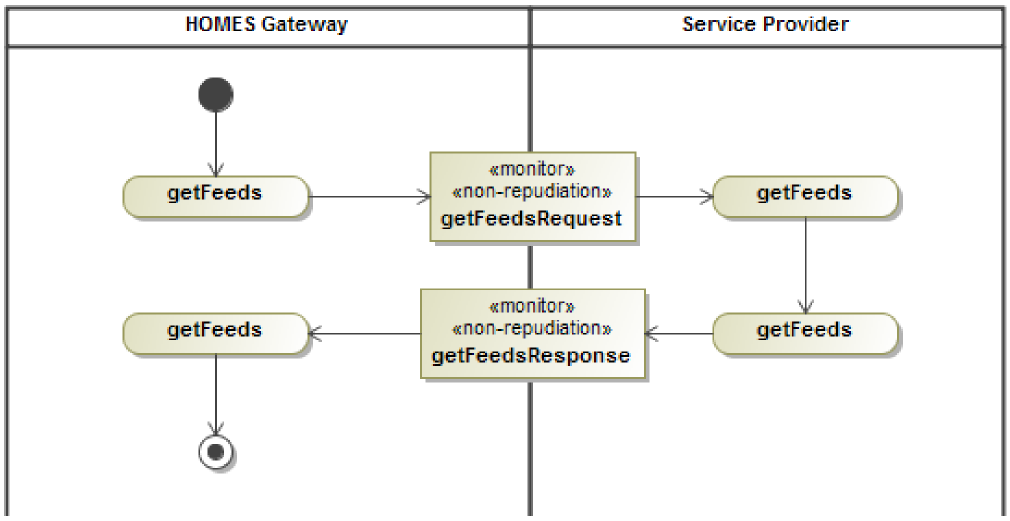


Figure 19 Workflow Model enhanced with security requirements

The next model that serves as input for the transformation process is the so called interface model. Basically, this is a class diagram defining services used within the business workflow. Operations of the services have to be named according to the activities defined in the workflow model (activity diagram). Figure 20 illustrates the interface model of our use case scenario.

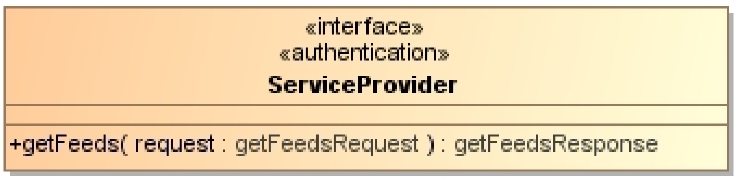


Figure 20 Interface Model enhanced with security requirements

Here one can see that the authentication requirement has to be assigned to the interface. Users do have to authenticate for the service itself and not for a specific operation within that service. In this example it should be the case that the operations from the Service Provider can only be performed by authenticated users.

With the stereotype interface we label classes describing the operations of the services used within the business process.

In order to be able to decrease the granularity within the transformation another input model is necessary. The Document Model describes the body of the messages sent between the actors. Figure 21 illustrates the structures of the messages from our simple example.

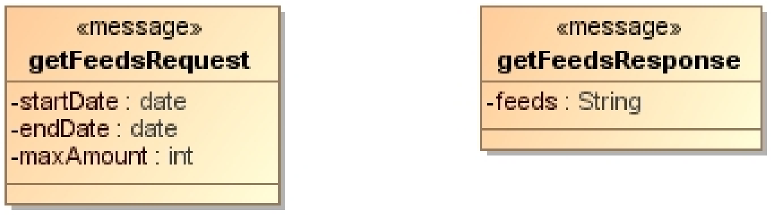


Figure 21 Document Model enhanced with security requirements

Using such class diagrams it is possible to attach security requirements to parts of data objects that flow from one actor to another. For example, only parts of the attributes may be subject to be signed electronically.

In order to highlight messages the stereotype <<message>> is used.

The security related aspects are defined in an UML profile. This profile (Figure 22) supports the user with the predefined stereotypes that are introduced throughout this section.

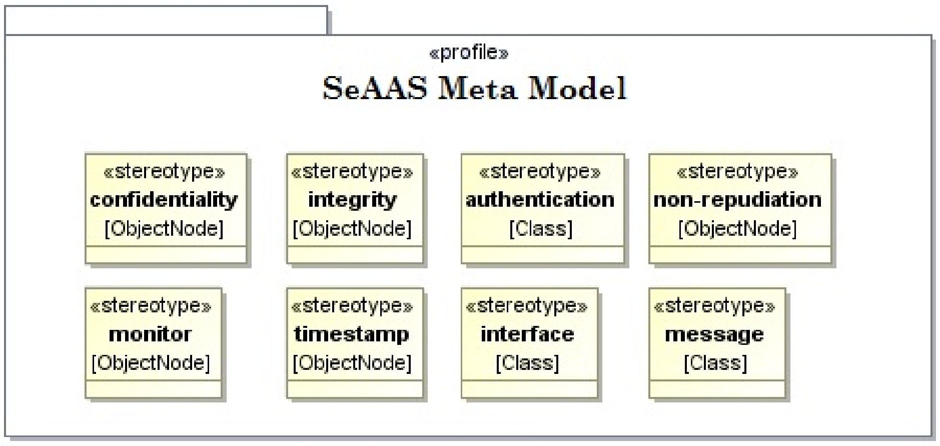

«profile»
SeAAS Meta Model
«stereotype» confidentiality [ObjectNode]
«stereotype» integrity [ObjectNode]
«stereotype» authentication [Class]
«stereotype» non-repudiation [ObjectNode]
«stereotype» monitor [ObjectNode]
«stereotype» timestamp [ObjectNode]
«stereotype» interface [Class]
«stereotype» message [Class]

Figure 22 The SeAAS profile

As we now have shown what the input to the transformation process is, we will present the outcome of it in the next section. Specifically we will illustrate the structure of the produced policy files.

4.1.2 Output Policies

When using the SeAAS concept there are basically three policies necessary to configure the security requirements. The first one configures the SeAAS engine itself. The second one defines which protocols (e.g. for non-repudiation) should be executed. The third policy is responsible to advise the security services with technical information like which element to encrypt. The generated file that represents the third policy has the shape of a java properties file. All policy files are stored within separate folders for each actor within the business process.

It is not possible to extract all information that is necessary for creating those policies just out of the input models. Technical details, e.g., what algorithms to use for encryption, are added during the transformation process which will be presented in the upcoming section.

4.1.3 Transformation

The transformation is responsible for generating the desired policies out of the input models. This transformation process consists of two refinement steps, where a Security Engineer is able to provide additional information. So the high level security requirements in the input models are getting refined to concrete policies. This refinement consists of two main steps which can be seen in Figure 23.

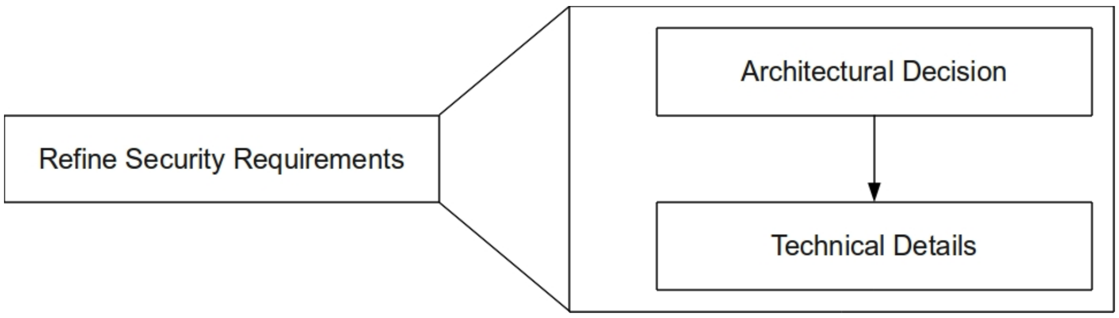


Figure 23 Steps within the transformation process

The “Architectural Decision” is the first refinement step, in which the decision for a specific protocol for example has to be made. The “Technical Details” refinement step provides the information required by the protocol which was chosen during the first refinement step.

4.1.3.1 Architectural Decision

During this refinement step an architectural decision has to be taken. That means that a Security Expert or Security Engineer has to choose a specific architectural protocol or pattern for each security requirement and message. When making this refinement the user is provided with a list of available patterns where one pattern for example corresponds to one protocol. As an example, during the refinement of the Authentication security requirement, the brokered authentication protocol is selected, and as a result a policy file, shown in Listing 2, is generated. This file asserts which protocol (line 6) to use. The Application Administrator then has to deploy this policy file on the SeAAS architecture. Specifically he has to deploy it within the PDP at the Operators side[8].

[8] In future versions the deployment can be automatized.

```
<wsp : Policy >
   <wsp : ExactlyOne >
     <wsp : All >
        <wsp : Policy >
          <ath : AthProtocolType xmlns : ath = `` http : // sectissimo . info / services /
               security / ath / protocoltype `` >
             <ath : BrokeredAuthProtocol / >
          < / ath : AthProtocolType >
        < / wsp : Policy >
     < / wsp : All >
   < / wsp : ExactlyOne >
< / wsp : Policy >
```

Listing 2 Generic Authentication Policy

Figure 24 shows the authentication patterns which are available in the prototype. That means as soon as there is the security requirement Authentication assigned to a service within the interface model the Security Engineer is able to choose one of the patterns from Figure 24.

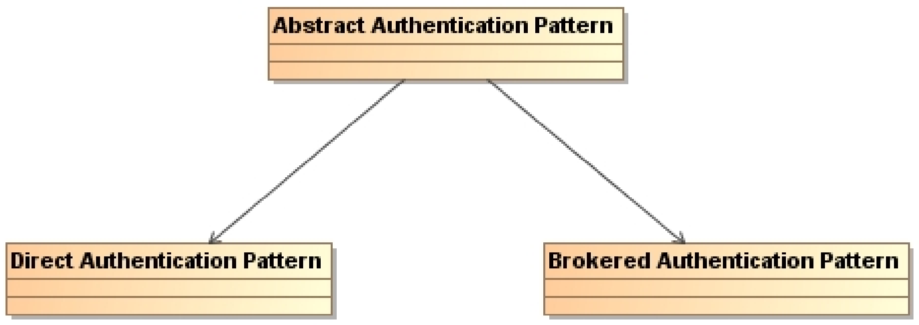

Abstract Authentication Pattern
Direct Authentication Pattern
Brokered Authentication Pattern

Figure 24 Available patterns for the authentication requirement

In this case the decision has to be done once per service and not per message, as the security requirement Authentication is an exception and only assigned to a whole service and not to a message within one interaction.

Once e.g. the brokered authentication pattern is chosen the SeAAS engine uses the protocol specified in the sequence diagram in Figure 25. After the SeAAS engine of the service requester's domain processed the policy, a token is requested from the identity provider. Once the request is successfully validated, the identity provider sends a token back to the service requester's domain. The SeAAS engine appends this token to the actual service request and sends it to the service provider. After the service provider successfully validated the token, the request is processed and the result sent back to the requester.

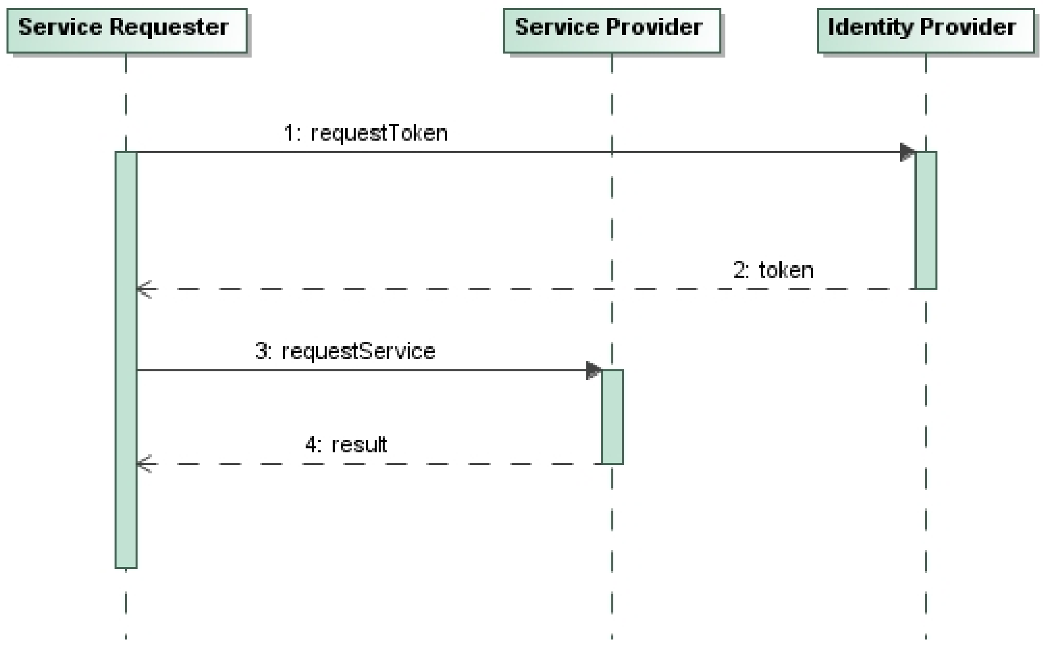


Figure 25 The brokered authentication protocol implemented by the SeAAS engine

After the decision for a specific pattern was made the next step is to provide the technical details to the transformation process. Using brokered authentication such information would be the location of the Identity Provider.

4.1.3.2 Technical Details

The second refinement step is about technical details. Here all additional information that could not have been saved within the input models is provided in order to generate the policies. Again, taking the brokered authentication example from above the following information has to be supplied:

- policy of the external identity provider,
- location of the identity provider,
- type of token.

The policy file of the external identity provider is the first information that is necessary in order to execute the brokered authentication protocol. This policy specifies the security requirements of the external identity provider. Furthermore, the locations of the identity provider as well as the type of the token (e.g. SAMLV1.1) have to be given. This way it is easily possible to replace one identity provider by another.

4.2 Prototype

In this section we will present the prototypical implementation of the concepts presented in Section 4.1.

The first section of this chapter introduces the graphical user interface. We then elaborate the meta-models. This is followed by details about the transformation process and presents the process of generating the actual code. The last section finally discusses the extensibility of the prototype.

4.2.1 Graphical User Interface

We start with the presentation of the graphical user interface of our prototype. The prototype is implemented as an Eclipse plug-in. The GUI basically consists of two parts: an input screen for importing the models and one for performing the refinement.

4.2.1.1 Working Models

Our framework leverages an approach based on Model Driven Development. Therefore it is necessary to have a set of underlying meta-models. For our prototype we made the decision to use Ecore as our meta-modeling language because our prototype is an Eclipse plug-in and Ecore is well integrated into Eclipse. Ecore is a component of the Eclipse Modeling Framework[9]. As the transformation process consists of two refinement steps we also have two categories of meta models which we now discuss further.

The first step, using our prototype, is to import the UML diagrams. During this process the information that is modeled within these diagrams is transformed into a single model. This model conforms to our *SecureChange-MDS-Meta-Model* (Figure 26).

The root eclass of this meta-model is *MDSMetaModel*. This eclass has a containment relation to the following three eclasses:

- *Type* - responsible to save the structure of the messages,
- *Partner* - saves all involved partners,
- *Service* - models the services.

An instance of the *MDSMetaModel* can model an unlimited number of services. The eclass *Service* has the attributes *name*, *namespace* and *prefix*. It also has a containment relation to the eclass *ServicePolicy* which again has a containment relation to the eclass *Authentication*. This states that the security requirement *authentication* can be attached to a service. Additionally the *Service* has a containment relation to the eclass *Operation*. This eclass has a single attribute *name* and containment relations to the eclasses *Request* and *Response*. Both of these eclasses have a reference to the eclass *Type*. This means that a service can have an unlimited number of operations where each operation has a request type and can have a response type. Both eclasses *Request* and *Response* have a containment relation to the eclass *MessagePolicy* which again has a containment relation to the eclasses *Monitor*, *Timestamp*, *Integrity*, *Confidentiality* and *NonRepudiation*, respectively. This states that these security requirements can be applied to the request and response message of an operation separately. Additionally the eclass *MessagePolicy* has two references to the eclass *Partner*. This way it is possible to store the sending and receiving partner within the message policy. This information is necessary to generate the policy files for the right partners.

[9] http://www.eclipse.org/modeling/emf/

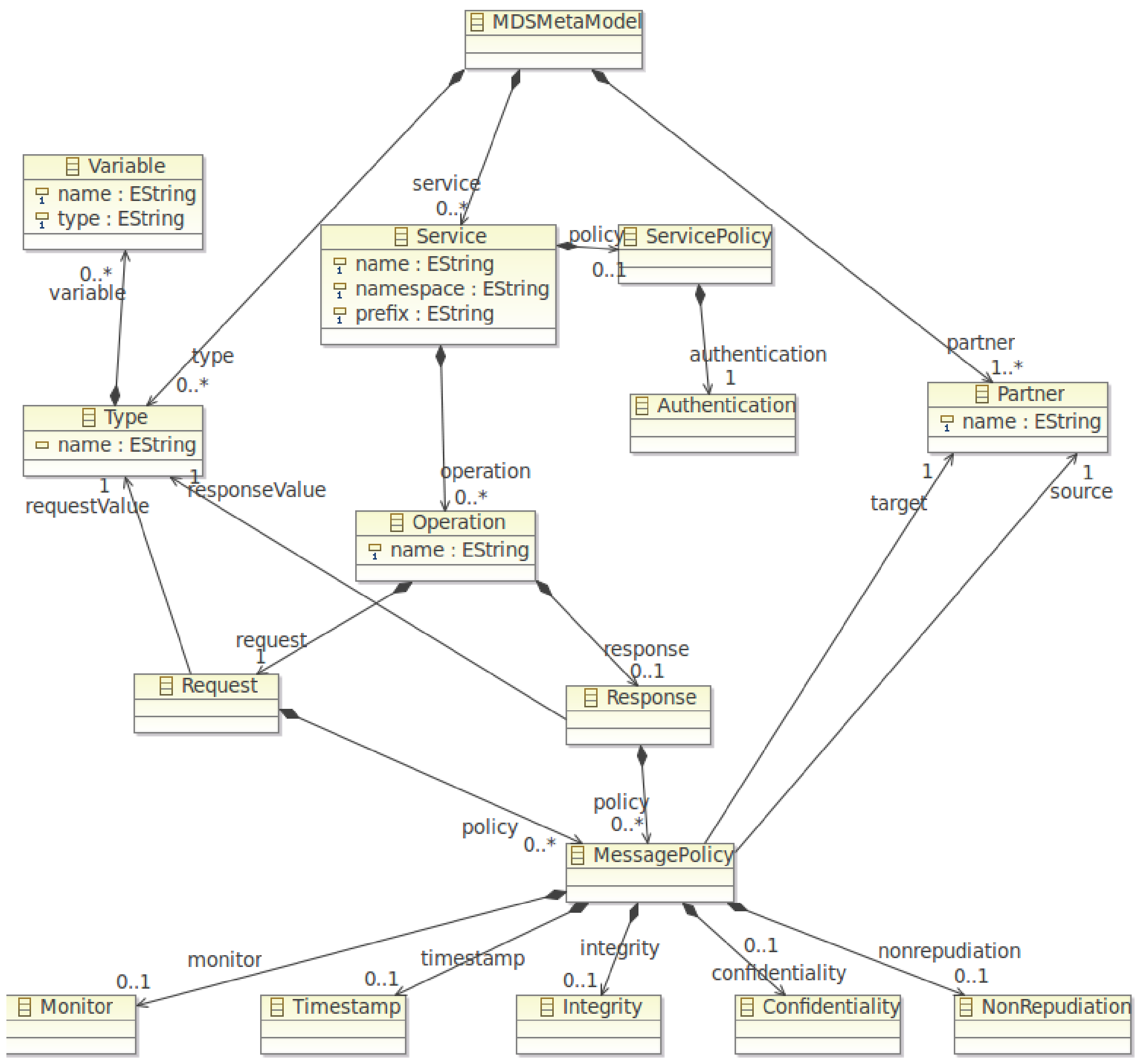


Figure 26 The SecureChange-MDS-Meta-Model

The reason why we transform the UML input models into an instance of this meta-model is that it becomes easier to retrieve and work with the information stored in this model.

The Atlas Transformation Language[10] is used for this transformation.

4.2.1.1.1 Decision Models

The policy generation process requires additional information. For that purpose the following meta-models are provided. We call the models used within the transformation process Decision Models.

[10] http://www.eclipse.org/atl/

Architectural Decision

The architectural decision is the first refinement step. For each security requirement a meta-model has been created. Within these meta models the available patterns are modeled.

Figure 27 illustrates the available patterns for the security requirement *monitor*. The ecore class *Monitor* is abstract and has a single attribute called *isRequired*. This attribute states that this eclass is necessary for the transformation. The available patterns for monitoring are the non-abstract subclasses of the class *Monitor*.

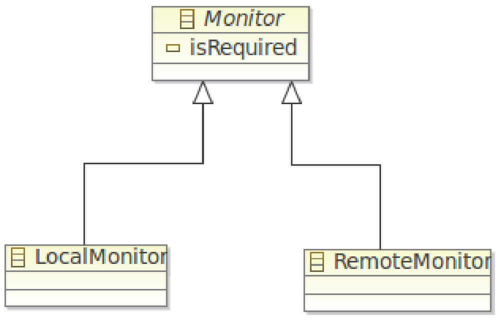


Figure 27 The meta-model for the monitor security requirement

Technical Details

The second kind of meta-models specifies the information necessary for the realization of each pattern. Therefore a meta-model for each of the available patterns is provided.

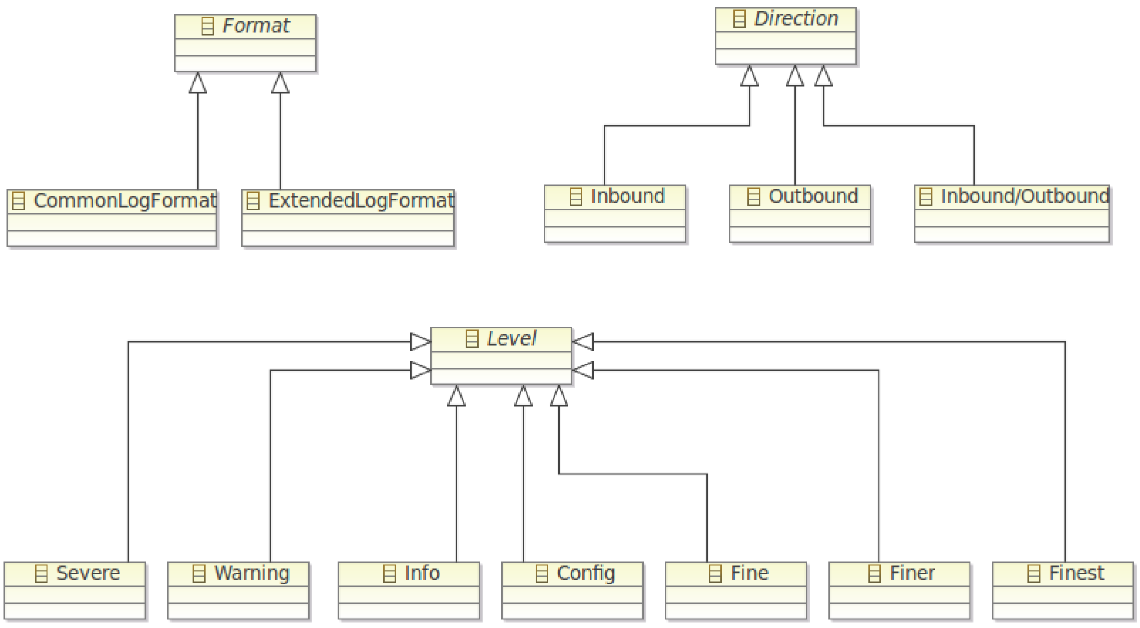


Figure 28 The meta-model for the local monitor pattern

Figure 28 shows the meta-model for the local monitor. All information necessary to generate the related policy file is modeled within this meta-model. This includes the format of the log file modeled by the abstract eclass *Format*. The supported patterns for the formats of the log files are defined using the non-abstract sub eclasses *CommonLogFormat* and *ExtendedLogFormat*. Concerning the level of granularity the Security Engineer can either choose *Inbound*, *Outbound* or *InboundOutbound*.

4.2.1.2 UML Model Import

This section presents the GUI component that enables importing UML models. The model import input screen is responsible to import the UML models which were created by the Domain Engineer. During this step the information stored within these models is transformed into a single model that conforms to our MDS-Meta-Model. We integrated this input screen into the standard Eclipse creation wizard which can be reached by File -> New -> Other… within Eclipse. The "SecureChange – Model Driven Security" import wizard is then located in an own folder ("SecureChange") as depicted in Figure 29.

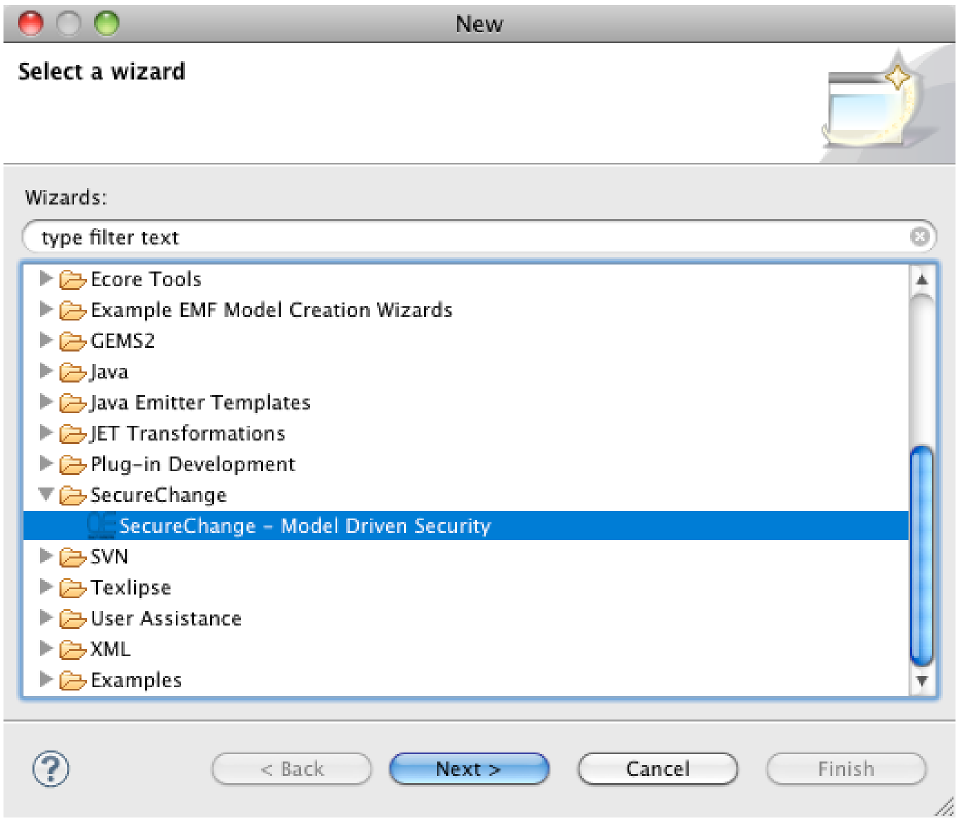
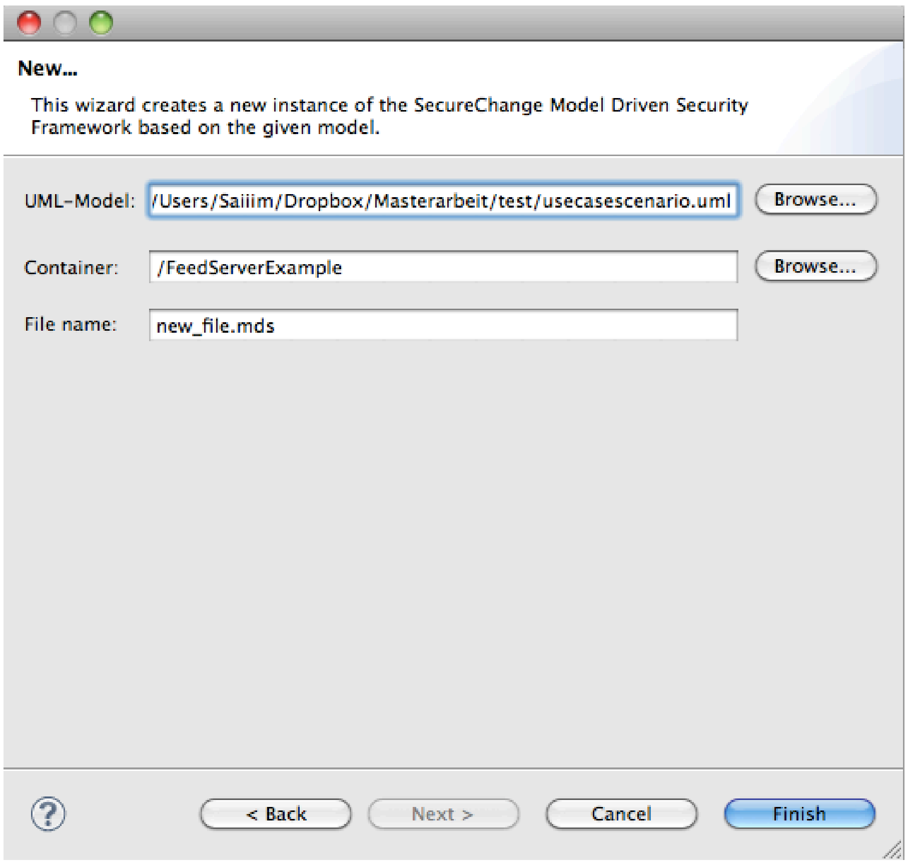


Figure 29 The integration of our import wizard within Eclipse

By choosing this wizard the actual input screen (Figure 30) is shown. There the Security Engineer has to specify three items:

- UML-Model - this field points to the UML file that stores the input models,
- Container - a project (or folder) has to be selected that will subsequently save the generated policies,
- File name - the name of the instance of the SecureChange-MDS-Meta-Model.


New...

This wizard creates a new instance of the SecureChange Model Driven Security Framework based on the given model.

UML-Model: /Users/Saiiim/Dropbox/Masterarbeit/test/usecasescenario.uml

Container: /FeedServerExample

File name: new_file.mds

Figure 30 The import screen

In order to import a model that was created with a UML Tool the file has to be exported as XMI file within the used modeling tool (e.g. MagicDraw[11]). This task has to be carried out by the Domain Engineer, who initially created these models. XMI is a standard created by the OMG to support tool independence and interoperability. When exporting the models from MagicDraw, beside the main file, that stores the information modeled by the Domain Expert, also files are created that are responsible for saving all UML profiles used within the model. It is necessary that all files are located within the same folder when the Security Engineer imports them using our framework.

During this import step the models are transformed into our SecureChange-MDS-Meta-Model. When finished, a new file is created in the selected container. Additionally, folders for each partner of the business process are generated. Subsequently the policy files will be stored separately within these folders.

4.2.1.3 Refinement Steps’ Transformation

After successfully having imported the UML models, the Security Engineer has to perform the refinements. The goal of these refinement steps is to provide additional information to the process of transforming the input models into policies.

[11] http://www.magicdraw.com/

The Security Engineer can choose the security requirement he wants to refine in the context menu illustrated in Figure 31. This context menu pops up when the Security Expert clicks with the right mouse button on the SecureChange-MDS-Meta-Model instance. Under the group “SecureChange” he can then choose a security requirement to refine.

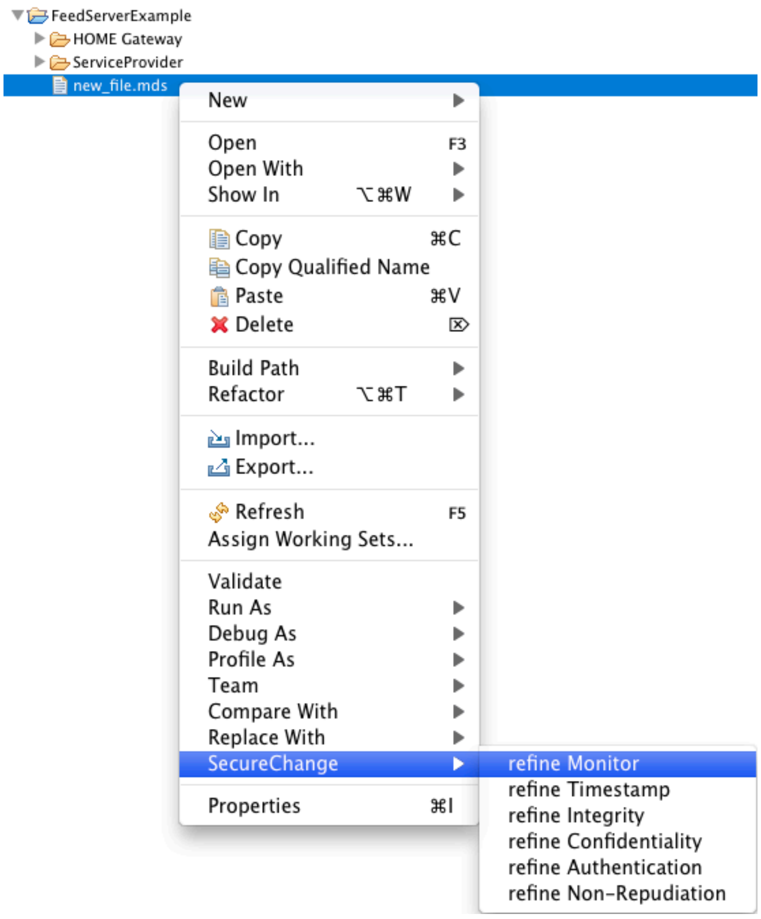
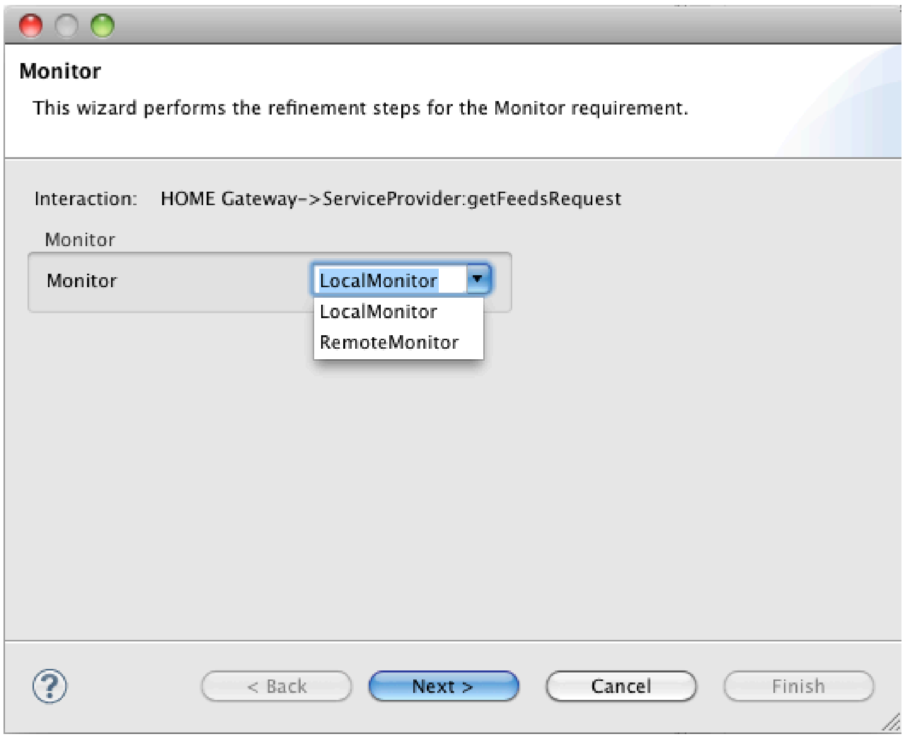


Figure 31 The context menu in eclipse with our plug-in

As already mentioned the first refinement step is making the decision for the architectural pattern. Therefore a wizard page is generated where the Security Expert has to choose one of the available patterns. Figure 32 shows this wizard page for Monitor requirement. This wizard pages are generated in an automatic way from the Decision Model that discussed before. The wizards consist of two pages. These pages slightly vary from one security requirement to another, but have the following core structure:

On the first wizard window the Security Engineer has to choose a specific pattern or protocol for the security requirement he is refining. On the second wizard window, he has to specify the technical details that are required for the pattern chosen on the first page.

Figure 32 illustrates the first window of the wizard that treats the monitoring requirement. This security requirement defines what and where a message should be logged. This window corresponds to the first refinement step, where the architectural pattern is chosen. Here the Security Expert is provided with the corresponding patterns that are available for this security requirement. In case of monitoring these two patterns are a local monitor and a remote one. This field is required. That means that a pattern has to be chosen in order to reach the next page of the wizard.



Monitor

This wizard performs the refinement steps for the Monitor requirement.

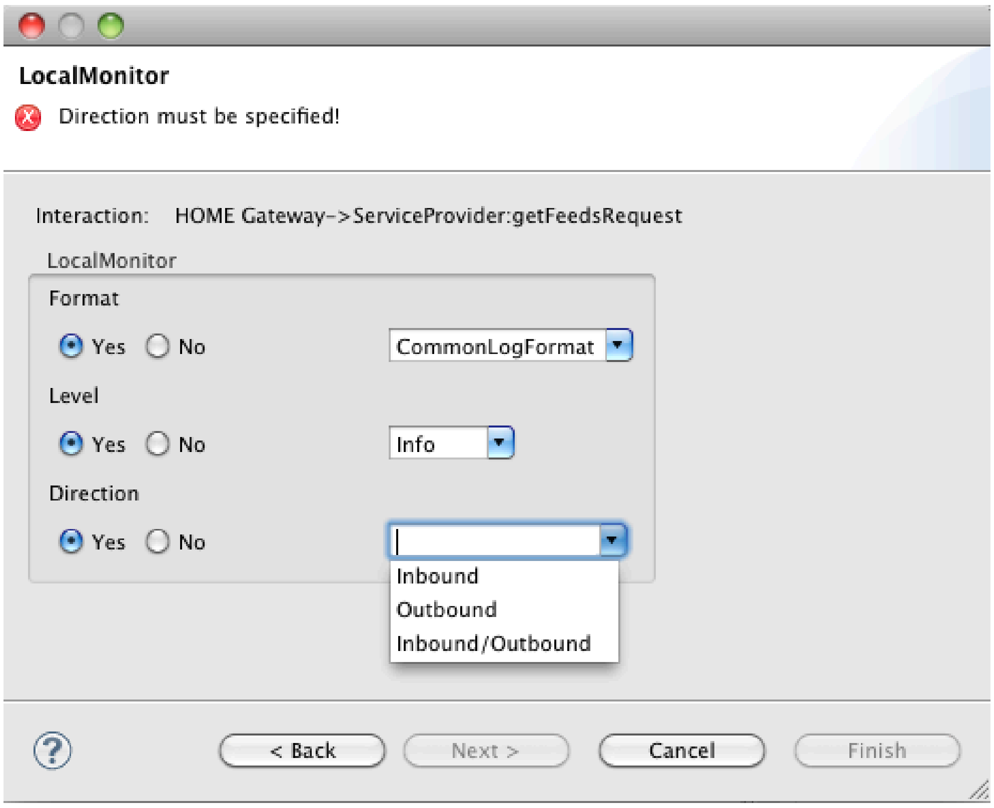
Interaction: HOME Gateway->ServiceProvider:getFeedsRequest

Monitor

Monitor: LocalMonitor

Figure 32 Wizard page for monitoring

After the Security Engineer made his decision, the next page in this wizard appears. Figure 33 illustrates the page that corresponds to the Local Monitor pattern. This page is responsible for the second refinement step. Here the Security Expert has to provide the technical details for the chosen pattern. Again, the Security Expert is supported with values to choose for the elements within this pattern. In this figure one can see that all elements are optional. They can be activated by selecting the corresponding radio button with the label “Yes”. Additionally we also implemented error detection. Basically this is necessary to check whether all required fields are filled out or not. In the latter case an error message is shown to the Security Engineer stating what has to be done in order to be able to get to the next page. An error does not only occur if a required element is not filled out but also when the radio button with label Yes is selected but the field is still empty, as it is the case in Figure 33.



LocalMonitor

Direction must be specified!

Interaction: HOME Gateway->ServiceProvider:getFeedsRequest

LocalMonitor

Format

[✓ Yes] No CommonLogFormat

Level

[✓ Yes] No Info

Direction

[✓ Yes] No

Inbound
Outbound
Inbound/Outbound

Figure 33 Wizard page for the local monitor

To summarize, there are two wizard pages for the security requirement Monitor. These two pages have to be filled out for each interaction where the security requirement is assigned. In the heading of the wizard pages (Figure 32 and Figure 33) one can see a string indicating the interaction the Security Engineer is refining with this page. This string consists of the source partner, target partner and the message.

Once the Security Engineer has entered all required information, the policy files are generated and saved within the folders of the actors which were created during the import phase.

At the end it has to be noted that the GUI (e.g. windows of Figure 32 and Figure 33) is generated based on the SecureChange-MDS-Meta-Model.

4.2.1.4 Policy Generation

The last step our prototype supports is the generation of the policy files as well as the properties files which will subsequently configure the SeAAS engines. We decided to use Java Emitter Templates for this task. We defined a template for each policy that can be generated using our prototype. Essentially there is a template for each of the meta-models that define the second refinement step (technical details). Listing 3 illustrates such a JET template. For practical reasons the namespaces are omitted. The shown template is responsible to generate the policy file for the fair non-repudiation protocol. Actually a java properties file is generated with the help of this JET template. This file is then used by the SeAAS engine to enforce the security requirement.

```
<%@ jet package="eu.securechange.homes.mds.transformation" imports="java.
    util.HashMap" class="FairNonRepudiation" %>
<%
  HashMap input = (HashMap)argument;
%>
trustedthirdparty = <%=input.get("TrustedThirdParty")%>
<%
    if(input.containsKey("SignedElements")) {
%>
signedelement = <%=input.get("SignedElements")%>
<%
    }
%>
```

Listing 3 The JET template for fair non-repudiation

Basically these templates define the static structure of the resulting policy file. Dynamic information like which element to encrypt is provided with a HashMap. This stores the values chosen or entered by the Security Expert. The keys of the HashMap again correspond exactly to the abstract eclasses from the corresponding meta-model.

4.2.2 Extensibility

The building blocks of our prototype are the meta-models. They are responsible to generate the graphical user interface and forward the information to the JET templates. In order to extend our framework the main part is to adapt these files to the own needs. Suppose the Framework Engineer wants to add a third non-repudiation protocol to the architecture. In a first step he has to add another subclass to the abstract eclass NonRepudiation in the meta-model. Then he has to create a new meta-model describing the details of this new protocol following the conventions. Corresponding to this meta-model and the requirements of his protocol he then has to create a JET template to actually generate the required policy file. Additionally he also has to change the JET template that creates the policy file for the generic non-repudiation service. There he simply has to add the option for his new protocol. As a last step he has to implement the actual protocol and deploy it on the target architecture.

That way, based on its service oriented conceptualization and its model driven interface our SeAAS framework provides substantial support for the evolution and maintenance of the security architecture.

5 Evaluation and Impact

In Year 3 the methodology of change driven security engineering was evaluated on the basis of the ATM case study. In terms of tools the MoVE tool was evaluated on the ATM case study and the HOMES case study, whereas the Security-as-a-Service-Architecture and its model-driven configuration interface were evaluated on the HOMES case study. In this section we shortly summarize the collected evaluation feedback and how it impacted our research and the development of the tools. The Section concludes with an outline of the impact of the research results of Work Package 2 from an industrial and scientific point of view.

5.1 Evaluation feedback

We have collected valuable feedback and also constructive criticism from the practitioners participating in the evaluation which was partly addressed already in our research and the prototypical tool development.

Change-driven security engineering process: The practitioners deem the methodology of change-driven security engineering capable to support, capture and analyze changes and evolutions in a complex domain such as ATM. The question to what degree complex security problems manifesting on different levels of abstraction could be handled would need to be tested in a more complex case study. In principle the approach is able to handle such problems in the required level of detail. The experts also believe that following our approach we are able to capture organizational settings and operational procedures of the ATM domain.

A point of criticism was related to the fact, that it is not always clear how all the artifacts and steps of the change-driven security engineering process are linked to each other. This highlighted the need for making the impact of changes transparent to the stakeholders following our methodology. This feedback lead us to design solutions for the MoVE tool which supports change-driven engineering processes which better highlight the consequences of each change handling steps to the respective users. Also clear instructions of the assigned tasks and activities which are expected to be executed by the respective stakeholder have to be communicated clearly. We have developed a basic design of a task pane which will be developed as part of future work and research.

In general, the methodology is deemed applicable in the ATM domain; however it clearly needs more integration and further evolution to be applicable in real productive environments for industrial usages.

MoVE tool: The industrial practitioners participating in the September workshop of a live session demonstrating the application of the MoVE tool in the ATM domain have also provided valuable feedback for improvements. As already highlighted in the comments related to change-driven security engineering a request on better highlighting state changes from on step to the other was made. We have already addressed this issue by implementing a summary of the state changes after each commit to the MoVE tool. However, we are aware that using clear visualizations for highlighting the impacts of changes in the respective models would improve the user experience. This is left open for future work and was not addressed in the SecureChange project.

An important question was raised by one industrial expert who was interested in how the state machines can be defined to adapt the tool to a proprietary industrial process. At the moment the underlying state machines are defined using SCXML. This manual definition is error-prone and we have not developed a user-friendly front-end for defining and checking the correctness of state machines. This is a very important point, which we will address as part of future work on the MoVE tool.

Validation feedback also pointed towards the need to improve and better support communication between stakeholders. We are aware of this point and in another research project an important objective will be the development of a task notification system on top of the MoVE tool. This task list informs stakeholders on their next tasks, the state of the models they are working on and the state of the overall process.

A feature that is already present in the MoVE tool and that was evaluated as highly important is the ability to provide evidence in the form of logs. Since every change committed by a stakeholder to the MoVE tool is analyzed and recorded we can provide extensive audit logs of who applied what changes during an overall change handling process.

Security-as-a-Service Architecture (SeAAS):

We have conducted several performance experiments of the SeAAS. The results of these performance experiments are documented in a technical report (cf. [31]). In the experiment the impact on the system performance of different Service Oriented Architecture (SOA) security approaches was studied.

It is most likely impossible to conduct such a study in a real SOA. Therefore we introduced and explained the development, execution and analysis of an experiment to compare the performance of the three possible SOA security approaches. For this, we designed a reference scenario based on SOA and Web services technologies in a first step. The scenario consisted of a service provider, an Enterprise Service Bus as communication backbone and the benchmark service consumer. In a second step we set up three alternative security architectures which were the actual target of evaluation:

- Endpoint Security (EndSec),
- Enterprise Service Bus Security (ESBSec),
- Security as a Service (SeAAS).

By utilizing various different global performance metrics, which all equivalently reacted to the studied factor, it was possible to conclude that there exists the following statistically significant difference in the performance of the system under test: SeAAS < EndSec < ESBSec. This means the execution time of SeAAS was lower than that of EndSec and ESBSec, thus having an overall better performance.

This implies implementing basic security functionality with a SeAAS infrastructure seems to be a feasible way. Based on this experimental setup, the developed tools and the gained knowledge several additional experiments with increasing complexity (e.g.

variations in the security mechanisms, etc.) should provide more details in this research area.

5.2 Impact

The research results and the tools developed in Work Package 2 have been presented in a number of scientific publications and to various industrial partners. In terms of scientific impact our goal was the dissemination of our research results on international software engineering conferences and in related journals.

For Work Package 2 the following publications about the results were published:

- Integrated Process (3 conferences, 1 journal, 1 professional journal),
- SeAAS (2 conferences, 1 journal),
- MoVE (2 conferences, 1 journal).

In terms of scientific impact we are also happy to report that the MoVE tool was presented to the project partners of the FP7 project PoSecCo. The project consortium decided to use the MoVE tool as an infrastructure to integrate the artefacts developed in the PoSecCo project (IT landscape models, business level policies, IT level security policies) and to support a change-driven process.

We have gathered strong interest from different industrial partners to whom we presented the MoVE tool. Three industrial partners are currently evaluating the feasibility of integrating it with their industrial tools.

6 Conclusion

In this deliverable we presented an overview of the completed implementation of the prototypical tool support which was the major focus of the third year for Work Package 2. In particular we completed on one hand the development of the MoVE tool implementation. On the other hand we implemented a model driven interface for the Security-as-a-Service-Architecture (SeAAS) to support a high-level configuration of the security services (SeAAS-MDS). Substantial effort was also invested in Year 3 for validation and evaluation. We applied the MoVE tool on the HOMES and the ATM case study and we have integrated the Security-as-a-Service-Architecture (SeAAS) in the HOMES gateway and delivered this prototypical implementation to our industrial partner for evaluation.

The deliverable started with a general discussion of the position of the various SecureChange tools with regard to the Integrated Process. Then the HOMES case and the change handling supported by MoVE were presented in a detailed walk-through outlining the underlying models and dynamics.

The next section focused on the MoVE tool implementation. We used the MoVE tool to support a fine-grained change driven security engineering process on the ATM case study. On the HOMES case study we supported a coarse-grained change driven process to connect three different tools with specific adaptors. The MoVE tool applications on the case studies were the basis for the validation and evaluation efforts. This deliverable contained a short summary of the architectural description of the MoVE tool before outlining the important aspects of extendibility and configuration.

The model driven security interface for the Security-as-as-Service Architecture (SeAAS-MDS) was described in detail in the next section. We outlined the input models and output policies and the transformation process to provide an overview of the framework. In addition the prototypical implementation was presented, focusing in particular on the graphical user interface and example models. The section on SeAAS-MDS concluded with a discussion on the extensibility of the MoVE tool.

The deliverables concludes with a discussion of the evaluation feedback and the impact of our research results. The research results and the tools developed in Work Package 2 have been presented in a number of scientific publications and to various industrial partners. In addition we have gathered positive feedback from the research community. The project consortium of the FP7 project PoSecCo decided to use the MoVE tool as an infrastructure to integrate different artefacts developed in the PoSecCo project and to support a change-driven process. Three industrial partners also showed strong interest and are currently evaluating the feasibility of integrating MoVE with their industrial tools.

7 Glossary

ADS-B: Automatic Dependent Surveillance Broadcast.

Artefact: distinguishes on an abstract level the different models and concepts which are used by the different Work Packages.

Asset: Something to which a party assigns value and hence for which the party requires protection.

ATM: Air Traffic Management.

API: Application Programming Interface.

CARiSMA: Enables to perform compliance analyses, risk analyses, and security analyses of software models.

Change request: A general description of some change in the system.

Change driven software engineering 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.

Consequence: The impact of an unwanted incident on an asset in terms of harm or reduced asset value.

CORAS: A method for conducting security risk analysis. The CORAS method provides a computerised tool designed to support documenting, maintaining and reporting analysis results through risk modelling.

CSV: Comma-separated values.

DOSGi: Distributed Open Services Gateway initiative.

EMF: Eclipse Modeling Framework.

EVe-TCF: EVe-TCF contains two main parts: off-device tools (executables than run on standard PC on Windows/Linux/Mac) written in portable C and on-device code to be integrated in smart card JavaCard virtual machine to verify control flow policies embedded in JavaCard packages at loading-time.

GUI: Graphical User Interface.

Integrated SecureChange Process: A light-weight change driven software process allows the application of different methodological approaches developed within the SecureChange project.

JET: Java Emitter Templates.

Likelihood: The frequency or probability of something to occur.

Living Models: A novel paradigm of model–based development, management and operation of evolving service oriented systems.

Living Security Engineering Process: A fully change driven security engineering process.

MACs: Message Authentication Codes.

Model Element States reflect the milestones in the lifecycle of the modelled artefacts and are used to reflect relevant changes.

MoVE (Model Model Versioning and Evolution): A tailor-made model repository to support Living Models and change-driven process.

OCL: Object Constraint Language.

OSGi: Open Service Gateway Initiative.

PDP: Policy Decision Point.

PEP: Policy Enforcement Point.

Reference architecture, specific technical platform or infrastructure realised according to an architectural paradigm (e.g., Security as a Service).

Risk: A possibility that a particular threat will adversely impact an element of the system architecture by exploiting a particular vulnerability. It is characterized by the likelihood of the unwanted incident and its consequence for a specific asset.

SeAAS (Security as a Service) stands for an architectural paradigm. It defines an architectural blueprint that transposes the model of Software as a Service to the security domain.

SeAAS-MDS (Security-as-a-Service Model driven security): An interface for the SeAAS framework which is based on the principles of Model Driven Security.

SecMER: An Eclipse-based heterogeneous modeling environment for managing evolving requirements models.

SECTET: A framework for model driven security engineering in SOA.

SCXML (State Chart XML): State machine notation for control abstraction.

SOA: Service oriented architecture.

SxC: Security-by-contract.

Threat: A potential cause of an unwanted incident.

Treatment category: A general approach to treating risks; the categories are avoid, reduce consequence, reduce likelihood, transfer and retain.

Treatment scenario: The implementation, operationalization or execution of appropriate measures to reduce risk level.

UML: Unified Modeling Language.

Unwanted incident: An event that harms or reduces the value of an asset.

VeriFast: A verifier for single-threaded and multithreaded C and Java programs annotated with preconditions and postconditions written in separation logic.

Vulnerability: A weakness, flaw or deficiency that opens for, or may be exploited by, a threat to cause harm to or reduce the value of an asset.

XMI: XML Metadata Interchange.

XML: Extended Markup Language.

8 Bibliography

[1] http://tools.ietf.org/html/rfc2753

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

[3] http://www.w3.org/TR/soap/

[4] http://www.osgi.org/

[5] http://cxf.apache.org/distributed-osgi.html

[6] http://www.bouncycastle.org/

[7] http://hadoop.apache.org/zookeeper/

[8] http://abdera.apache.org/

[9] http://ws.apache.org/axiom/

[10] http://xml.apache.org/xalan-j/

[11] http://ws.apache.org/commons/neethi/

[12] http://santuario.apache.org/

[13] http://www.osgi.org/download/r4-v4.2-cmpn-draft-20090310.pdf

[14] http://www.w3.org/Submission/WS-Policy/

[15] http://www.magicdraw.com/

[16] http://www.eclipse.org/

[17] http://help.eclipse.org/indigo/index.jsp

[18] http://move.q-e.at/

[19] http://subversion.apache.org/

[20] Bill Burke. RESTful Java with JAX-RS. O'Reilly, 2010.

[21] D. Steinberg, F. Budinsky, M. Paternostro, E. EMF: Eclipse Modeling Framework, 2nd Edition, Published Dec 16, Addison-Wesley Professional, 2008.

[22] OMG, OMG Unfied Modeling Language (OMG UML) Superstructure V2.4.1, http://www.omg.org/spec/UML/2.4.1/

[23] OCL, OMG, Object Constraint Language(OCL) Specification V2.2, http://www.omg.org/spec/OCL/2.2

[24] R. S. Hall, K. Pauls, S. McCulloch and D. Savage. OSGi in Action Creating Modular Applications in Java. Manning Publications, 2010.

[25] Peter Kriens and BJ Hargrave. Listeners Considered Harmful: The "Whiteboard" Pattern. OSGi Alliance, 2004.

[26] Jianying Zhou and D. Gollman. A Fair Non-repudiation Protocol. IEEE Symposium on Security and Privacy, pages 55-61, 1996

[27] Blake Dournaee. XML Security. McGraw-Hill, 2002.

[28] Christian Connert, Benchmarking SOA Security Approaches, Master thesis 2010.

[29] Hafner, M. and Breu, R. Security Engineering for Service-Oriented Architectures. Springer Verlag GmbH, Berlin/Heidelberg. 2008.

[30] R. Breu, M. Memon, F. Innerhofer-Oberperfler, M. Weitlaner, M. Breu, M. Hafner, R. Scandariato, K. Yskout, K. Buyens, B. Fontan, F. Paci, E. Chiarani. An architectural blueprint and a software development process for security-critical lifelong systems. Deliverable D.2.1, SecureChange, 2010.

[31] F. Innerhofer-Oberperfler, S. Löw, R. Breu, M. Breu, M. Hafner, B. Agreiter, M. Felderer, P. Kalb, R. Scandariato, B. Solhaug. A configuration management process for lifelong adaptable systems. Deliverable D.2.2, SecureChange, 2011.

[32] C. Connert, S. Forster, M. Hafner. SeAAS – Introduction and Empirical Performance Evaluation, Technical Report QE-2011-24, 2011.