D.3.4 PROOF-OF-CONCEPT CASE TOOL

Michela Angeli (UNITN), Karmel Bekoutou (UNITN), Gábor Bergmann (BME), Elisa Chiarani (UNITN), Olivier Delande (THA), Edith Felix (THA), Fabio Massacci (UNITN), Bashar Nuseibeh (OU), Federica Paci (UNITN), Thein Tun (OU), Dániel Varró (BME), Koen Yskout (KUL), Yijun Yu (OU)

Document information

Document Number	D.3.4
Document Title	Proof-of-Concept CASE Tool
Version	2.12
Status	Final
Work Package	WP 3
Deliverable Type	Prototype
Contractual Date of Delivery	31 January 2012
Actual Date of Delivery	26 January 2012
Responsible Unit	BME
Contributors	OU, UNITN, BME, THA
Keyword List	requirements evolution, argumentation, evolution rules,
Dissemination level	PU

Document change record

Version	Date	Status	Author (Unit)	Description
0.1	7 September 2011	Draft	Gábor Bergmann (BME)	First Draft based on original Tool Readme
0.1	15 December 2010	Draft	Federica Paci (UNITN)	First Draft of Demo Scenario
0.2	17 December 2010	Draft	Gábor Bergmann (BME)	Started the executive summary. Elaborated details of each demo script, except for the argumentation Started Technical Overview
0.3	23 December 2010	Draft	Gábor Bergmann (BME)	Completed the executive summary. Improved demo scripts, added argumentation; attached screenshots Resolved issues raised before and during WP3 telephone conference Added Transformation to Technical Overview Added References
0.4	23 December 2010	Draft	Dániel Varró (BME)	Revised the executive summary.
0.5	24 December 2010	Draft	Gábor Bergmann (BME)	Added explanatory Figure to Section 2.
0.6	28 December	Draft	Gábor Bergmann	Added Figure to Ex.

	2010		(BME)	Summary.
0.7	12 January 2011	Draft	Karmel Bekoutou (UNITN)	Quality check completed-minor remarks
0.9	13 January 2011	Revised Draft	Gábor Bergmann (BME)	Updated Scenario descriptions to match reviewer comments.
1.0	14 January 2011	Revised Draft	Gábor Bergmann (BME)	Addressed all minor remarks
1.1	19 January 2011	Final	Gábor Bergmann (BME)	Modified future integrations plans in Executive summary, according to discussions in the General Assembly.
1.2	20 January 2011	Final	Gábor Bergmann (BME)	Fixed authors list.
2.0	07 December 2011	Y3 Draft	Gábor Bergmann (BME)	Integrated tool manual, added shift to Y3
2.1	19 December 2011	Y3 Draft	Gábor Bergmann (BME)	Restructured according to instructions from Fabio, moved to new Demo Corner example
2.2	20 December 2011	Y3 Draft	Federica Paci (UNITN)	Rephrased Exec. Summary, updated integration details
2.3	20 December 2011	Y3 Draft	Gábor Bergmann (BME)	Final touches
2.4	28 December 2011	Y3 Draft	Michela Angeli (UNITN)	first quality check completed-minor remarks
2.5	4 January 2012	Y3	Koen Yskout	Y3 Scientific review (first

		Draft	(KUL)	round)
2.6	16 January 2012	Y3 Draft	Gábor Bergmann (BME)	Addressing comments of internal review
2.7	19 January 2012	Y3 Draft	Koen Yskout (KUL)	Y3 Scientific review (second round)
2.8	19 January 2012	Y3 Draft	Michela Angeli (UNITN)	second quality check completed-minor remarks
2.9	23 January 2012	Y3 Draft	Olivier Delande, Jérôme Le Noir (BME)	Contributing industrial evaluation of EMF-INCQUERY
2.10	26 January 2012	Final	Gábor Bergmann (THA)	Addressing comments of second internal review
2.11	26 January 2012	Final	Michela Angeli (UNITN)	Third quality check completed-minor remarks
2.12	26 January 2012	Final	Gábor Bergmann (THA)	Addressing comments of third internal QC, finalizing URLs

Executive summary

This document is the description of the WP3 CASE tool prototype that was built to demonstrate the concepts and workflow of SeCMER, the SecureChange Methodology of Evolutionary Requirements.

The demonstrator is an Eclipse-based heterogeneous modeling environment for evolving requirements models that are formulated in different languages appropriate for different work phases and domain expertise. It also provides an analysis toolset to conduct interactive or automated, formal and informal security analysis. Deliverable D.3.4 consists of two components: a software prototype (called SeCMER Demonstrator) and this document that presents the research results, the demonstrator architecture and a sample case study.

The main results presented in the demonstrator (and thus in the current document) are the following:

- **Multi-aspect modeling** approach where the security requirements model is composed of several views in different modeling languages, and each work phase can use the facet of the model that is most appropriate to represent their tasks and domain expertise. To cope with the evolving nature of the model, changes made to any of the view models can be **incrementally synchronized** to all other views where appropriate using change-driven transformations.
- **Automated pattern-based analysis** for certain security properties. This analysis is resilient to change and the results are continuously and incrementally kept up-to-date.
- **Interactive and formal argumentation analysis** to support security experts in conducting arguments to verify security properties. Taking up the challenge of evolving models, these argumentation features are complemented by partially automated strategies to cope with changes to the requirements model.

During Year 3, we have explored the relationship between SeCMER and CORAS. We have mapped concepts in SeCMER and CORAS conceptual models that are used to represent assets that are critical for the achievement of organization's strategic objectives and means to protect assets in a cost-effective manner. The mappings have been expressed as VIATRA2 graph transformation rules. Due to being developed as one of the last results of the project, there was no time to integrate the mappings into the demonstrator to support synchronization between SeCMER and CORAS models under change; but this is of little consequence, since the prototype tool has already demonstrated the feasibility of similar model transformations.

We have also investigated the role that SeCMER methodology and tool can have in the industrial security process proposed by Thales as described in Appendix G. As for the EMF-INCQUERY component in particular, Thales has conducted an initial evaluation in an industrial context, as documented in Appendix C.

In addition to these activities, WP3 collaborated with Deep Blue to validate both artifacts and also the tool by the tentative application of the tool to the ATM Case Study for the collection, modeling and analysis of the evolution of security requirements in a

real industrial context and a final an evaluation workshop with ATM experts. The feedback from ATM experts is analyzed in detail in Appendix D.

The workshop feedback contributed to the improvement of the tool. Numerous changes were made to the tool at the request of the ATM experts, such as a customization of the Si* User Interface to fit SeCMER concepts and workflow. The changes include the following:

- an improved Graphical User Interface, with user-friendly features including wizards and export / import functionality;
- a customized adaptation of the Si* diagram features to be more aligned with SeCMER concepts;
- an extended mapping between Si* and SeCMER to enable Si*-based modeling for all of the concepts involved in the Year 2 and Year 3 demonstration scenarios;
- added new Security Patterns to cover more security issues, including violations of the least privilege principle;
- added support for generating a dynamic list of quick fixes for a single security violation;
- miscellaneous bug fixes.

The structure of the deliverable document is the following:

- The prototype tool is briefly introduced in Section 1.
- A walkthrough of some important features is presented in Section 2, in context of the ATM case study.
- Section 3 includes further discussion of the background, limitations and future of the SeCMER tool.
- Design and implementation details are provided in Appendix A.
- The User's Guide for the SeCMER tool is located in Appendix B.
- A report written by Thales of the initial industrial evaluation of the underlying technology EMF-INCQUERY is included as Appendix C
- The results of the workshop with ATM experts are reported in Appendix D.
- Scientific publications related to the SeCMER tool are attached as Appendix E, Appendix F and Appendix G.

Table of Contents

List of Figures

1 Introduction

The goal of the SeCMER tool is to support the Requirement Engineers in following the SeCMER methodology. The recommended way to capture a Requirements Model is in the Si* formalism, although other languages are supported as well. The requirements model can then undergo automated, pattern-based static analysis and manual, informal argumentation analysis to discover security issues.

The tool provides basic viewing and editing functionality to the integrated aspect models (Si*, abstract SeCMER model and Argument model currently). The basic modeling functionality includes the following:

- Using the Si* / Tropos visual language to identify e.g. stakeholders, resources provided, actions performed, functional and security goals stated and met, decomposition, input / output dependencies of actions or goals, trust between stakeholders and finally delegation of duties or access.
- Recording arguments carried out by security experts that identify security liabilities and problems. Also modeling the break down structure of arguments, the interrelation with counter-arguments (rebuttals, mitigations), and the back-tracing of elementary facts to concepts in the requirements model.

Additionally, the following added-value mechanisms are implemented:

- There is on-the-fly bi-directional synchronization between the SeCMER and Si* representation of requirement models. This means that the same requirements model is always represented as a visual Si*/Tropos diagram, and also as a more abstract underlying SeCMER model.
    - Changes made in the abstract EMF representations (like the tree editor and the GMF Tropos Diagram) are transformed and synchronized between the SeCMER and Si* aspects on the fly.
    - Textual formats (like the .ontology format of the SeCMER requirement model) are more detached: updates to and from them are only propagated upon saving. This is due to technical constraints in the prototype.
    - Problem Frames (OpenPF) models can be unidirectionally transformed into the .ontology format.
- An automated static security analysis considers a class of security problems that are defined by an extensible set of **security patterns**[1]. Each security

[1] Not to be confused with the notion of „security pattern" as in security-aware architecture/design patterns. Security patterns in context of SeCMER are patterns in the sense of pattern matching. In analogy to the terminology of design patterns, these security patterns would be considered *anti-patterns*.

pattern is a graph pattern or model query that identifies parts in the model that violate a given security property. The analysis is performed by Evolution Rules that detect violations of the given security properties and offer *automated solutions*. Violations appear as Eclipse problem markers (warnings). The suggested solutions appear as Quick Fix rules. The initial set of security patterns provided in the prototype are chosen according to these guidelines:

    - The main points of focus are **trust** (which can be explicitly modeled, and interpreted transitively), **access** (which can also be granted / delegated transitively), and **need** (expressed by carrying out an action that consumes a resource).
    - The security patterns only consider assets that are protected by security goals. If a resource is not a valuable asset, then no distracting problem reports will be generated e.g. in case of untrusted access.
    - Security violation reports can be suppressed by manual arguments supporting the satisfaction of the security goal.

  The security patterns detect the following problems:

    - If there is access to an asset without trust (regardless of need), then it is considered a violation of the trusted path property.
    - If there is access to an asset without the need thereof (regardless of trust), then it is considered a violation of the least privilege property.
    - If there is need for an asset but no actual access, then the model is reported as inconsistent / incomplete.

- Traceability links can be established between the argument and requirement models. They enable automatic detection of requirement changes that make a manually conducted argument obsolete. Model changes involving the ground facts may trigger a notification that alerts the user about the possibility that the argument may have become invalid due to the change. The security experts can then revisit these arguments to reflect the evolution, while no costly revision process is required for unaffected arguments.

The tool is designed to be extensible. Apart from the SeCMER requirements model, the Si* model and arguments, additional aspect models can also be joined in the future. There can be one-directional transformations, such as the one between OpenPF requirement models and SeCMER. However, the tool provides support for traceability maintenance (as in the case of arguments), and a framework for bidirectional live synchronization (demonstrated in the prototype tool by the SeCMER ↔ Si* mapping).

2 Feature Tour

We are going to illustrate a subset of features supported by WP3 CASE tool prototype using the Air Traffic Management (ATM) case study. The ATM case study will be used during the whole tour since it features requirements-based early security analysis.

2.1 Example Scenario from the ATM domain

This Section follows the common Demo Script of the ATM-oriented Work Packages of SecureChange.

2.1.1 Change Requirements (Organizational Level)

The introduction of the Arrival Management subsystem (AMAN) affects Controller Working Positions (CWPs) as well as the Area Control Center (ACC) environment as a whole. The main foreseen change in the ACC from an operational and organizational point of view is the automation of tasks (i.e. the usage of the AMAN for the computation of the Arrival Sequence) that in advance were carried out by Air Traffic Controllers (ATCOs), a major involvement of the ATCOs of the upstream Sectors in the management of the inbound traffic.

Goal: The ATCO system interfaces that provide access to actors with various roles and authorizations need to make information available only to authorized personnel or trusted systems, on a need-to-know basis.

2.1.2 Security Properties (Need-to-Know principle)

It is necessary to guarantee specific security properties after the deployment of the identified changes requirements. In particular, an operational need-to-know principle can be defined in terms of the following security properties:

Information Access. Authorized actors (or systems) must have access to confidential information regarding queue management in the terminal area. Access to information needs to comply with specific role-based access control rules drawn from the operational requirements.

Information Protection. Unauthorized actors (or systems) are not allowed to access confidential queue management information.

Information Need. Confidential queue management information can be accessed by authorized actors (or systems) only when the information is necessary for operational purposes, which may vary even in real time, due to particular conditions (bad weather, emergency status, etc.).

2.1.3 User Story

A new Arrival Management subsystem (AMAN) is introduced in the air traffic control system. Among other duties, the AMAN schedules the arrival of a State Flight, which is a highly sensitive flight with high-ranking state officials on board. Because of the unified electronic representation of flights required by AMAN, information regarding the State Flight must be handled by the ACC systems as part of the Flight Data in the after scenario.

ATCOs that are currently on duty in the ACC control room are aware of this sensitive information and take that into account while working on their sectors. The ATCO supervisor, among other goals, is responsible for the security of confidential information. Each operation CWP shows any relevant information about the flights. It is necessary to guarantee that confidential information becomes available to actors operating inside and outside the ACC control room only when information is necessary for achieving their operational goals. For instance, an external contractor's System Engineer who is authorized to access the control room and a CWP to perform system maintenance should not be allowed to access confidential information such as the State Flight.

The AMAN is linked to the information communication network that makes scheduled flight information available to other services. However, it is necessary to prevent the confidential State Flight information from becoming available to other actors operating outside the ACC control room. Therefore, actors like Service Engineers, who might provide services on ground, are allowed to access only non-confidential information about the State Flight.

2.2 Overview of demonstrated use cases

The audience of the tour will be able to see the following scenario steps, each of which is supported by the tool: The steps of the tour were chosen such that they follow a typical requirements evolution workflow, also featuring the contributions of WP3.

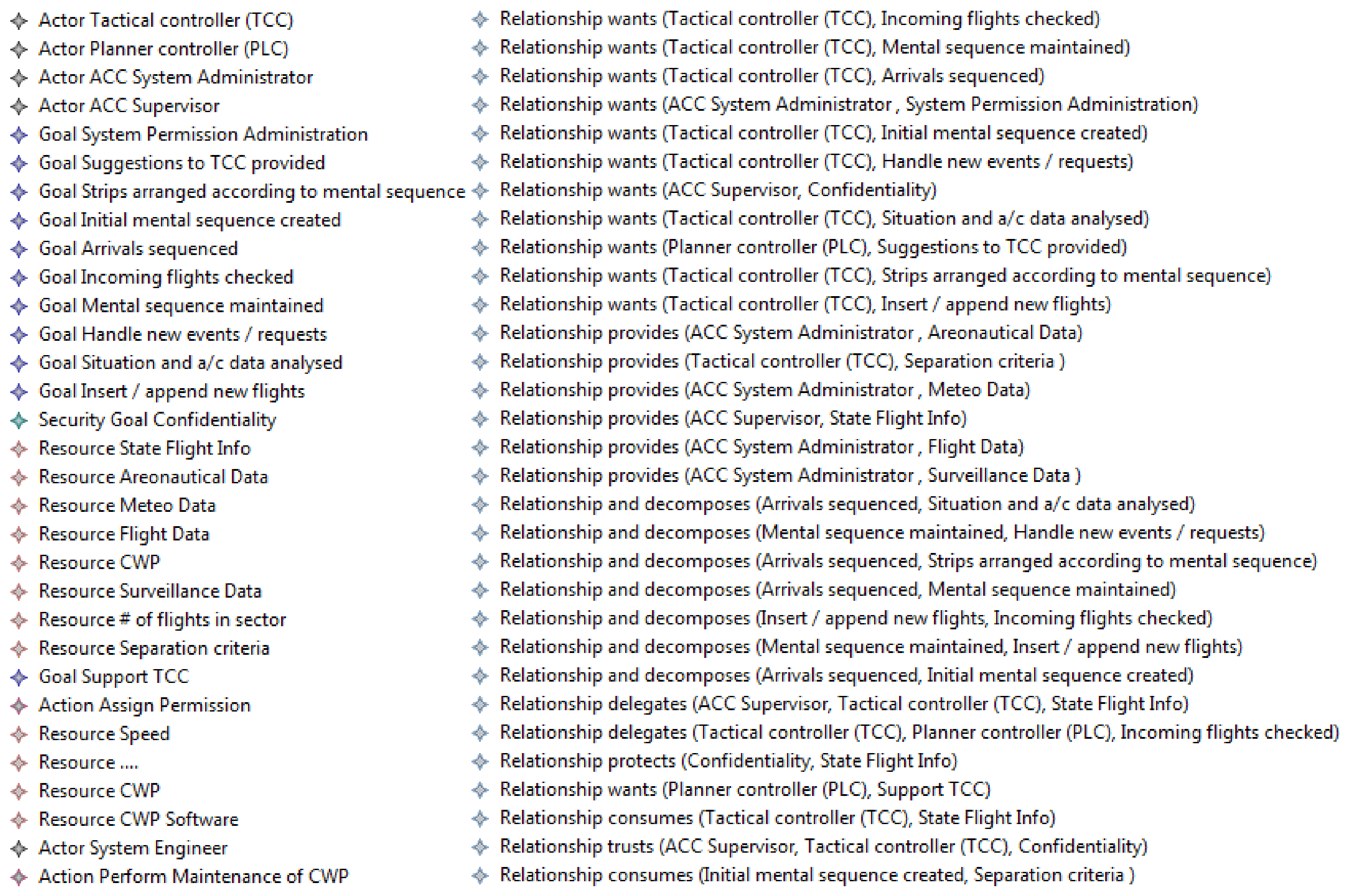
1) **Synchronization between models in different requirements formalisms.** We will illustrate that correspondence in the before state is preserved automatically in the after state, e.g. how the addition of a new actor "AMAN" in the SI* model showcasing the organizational level change (see Section 2.1.1) is reflected on the SeCMER model.
2) **Detection of a violation of information protection (confidentiality in particular) and automatic corrective action based on evolution rules.** We will show how the least privilege principle is violated for State Flight Info in the post-state (after state) SeCMER model because there is a security goal "Confidentiality of State Flight Info" protecting the asset State Flight Info; where the actor System Engineer has access to State Flight Info (as it is contained in Flight Data in the after state, which in turn is available via CWP), even though it does not require the asset for performing its operational duty. This step invokes an evolution rule where the event and condition represent a pattern matching such a situation, and where the action part suggests a corrective action (e.g. adding to the model a potential missing action that is performed by the System Engineer and consumes the asset).

3) **Argumentation for the least privilege (need-to-know) property**. We will show how argumentation analysis [5] can be carried out for the access control property applied to the Flight Data.

2.3 Synchronization between models in different requirements formalisms

According to SeCMER, a single requirement model can be composed of views in multiple modeling languages, based on the expertise of requirements engineers and the domain-specific style of modeling. The challenge lies in preserving consistency while correctly mapping modifications in a (source) modeling formalism to incremental changes in another (target) modeling language, especially on-the-fly as the modifications are being made. The following is a demonstration of incremental synchronization between views of the requirement model in different formalisms. The concept is demonstrated by the transformation between Si* and the SeCMER conceptual model in the ATM case study.

Initially, the two views in the model are consistent, reflecting the same model (the pre-state). See Figure 1 for an extract of the abstract SeCMER model and Figure 2 for its Si* visualisation. In this state, the actors include ACC Supervisor, Tactical Controller (TCC), ACC System Administrator, Planner Controller (PLC) and the System Engineer. The actors handle data resources including Flight Data (which aggregates Speed and other kinds of data), Meteo Data, etc., as well as the physical resource CWP (containing CWP Software and data resources), and delegate these to each other. These data resources are used as inputs of various actions and operational goals performed by the actors, such as the operation goal “Arrivals sequenced” or the action “Perform Maintenance on CWP”. The Actor ACC Supervisor also has a Security Goal “Confidentiality”, which entails the protection of the sensitive asset State Flight Info. All shown actors are trusted to comply with this Security Goal.



- Actor Tactical controller (TCC)
- Actor Planner controller (PLC)
- Actor ACC System Administrator
- Actor ACC Supervisor
- Goal System Permission Administration
- Goal Suggestions to TCC provided
- Goal Strips arranged according to mental sequence
- Goal Initial mental sequence created
- Goal Arrivals sequenced
- Goal Incoming flights checked
- Goal Mental sequence maintained
- Goal Handle new events / requests
- Goal Situation and a/c data analysed
- Goal Insert / append new flights
- Security Goal Confidentiality
- Resource State Flight Info
- Resource Areonautical Data
- Resource Meteo Data
- Resource Flight Data
- Resource CWP
- Resource Surveillance Data
- Resource # of flights in sector
- Resource Separation criteria
- Goal Support TCC
- Action Assign Permission
- Resource Speed
- Resource ....
- Resource CWP
- Resource CWP Software
- Actor System Engineer
- Action Perform Maintenance of CWP

- Relationship wants (Tactical controller (TCC), Incoming flights checked)
- Relationship wants (Tactical controller (TCC), Mental sequence maintained)
- Relationship wants (Tactical controller (TCC), Arrivals sequenced)
- Relationship wants (ACC System Administrator , System Permission Administration)
- Relationship wants (Tactical controller (TCC), Initial mental sequence created)
- Relationship wants (Tactical controller (TCC), Handle new events / requests)
- Relationship wants (ACC Supervisor, Confidentiality)
- Relationship wants (Tactical controller (TCC), Situation and a/c data analysed)
- Relationship wants (Planner controller (PLC), Suggestions to TCC provided)
- Relationship wants (Tactical controller (TCC), Strips arranged according to mental sequence)
- Relationship wants (Tactical controller (TCC), Insert / append new flights)
- Relationship provides (ACC System Administrator , Areonautical Data)
- Relationship provides (Tactical controller (TCC), Separation criteria)
- Relationship provides (ACC System Administrator , Meteo Data)
- Relationship provides (ACC Supervisor, State Flight Info)
- Relationship provides (ACC System Administrator , Flight Data)
- Relationship provides (ACC System Administrator , Surveillance Data)
- Relationship and decomposes (Arrivals sequenced, Situation and a/c data analysed)
- Relationship and decomposes (Mental sequence maintained, Handle new events / requests)
- Relationship and decomposes (Arrivals sequenced, Strips arranged according to mental sequence)
- Relationship and decomposes (Arrivals sequenced, Mental sequence maintained)
- Relationship and decomposes (Insert / append new flights, Incoming flights checked)
- Relationship and decomposes (Mental sequence maintained, Insert / append new flights)
- Relationship and decomposes (Arrivals sequenced, Initial mental sequence created)
- Relationship delegates (ACC Supervisor, Tactical controller (TCC), State Flight Info)
- Relationship delegates (Tactical controller (TCC), Planner controller (PLC), Incoming flights checked)
- Relationship protects (Confidentiality, State Flight Info)
- Relationship wants (Planner controller (PLC), Support TCC)
- Relationship consumes (Tactical controller (TCC), State Flight Info)
- Relationship trusts (ACC Supervisor, Tactical controller (TCC), Confidentiality)
- Relationship consumes (Initial mental sequence created, Separation criteria)

Figure 1. Evolution pre-state in abstract SeCMER model (extract)

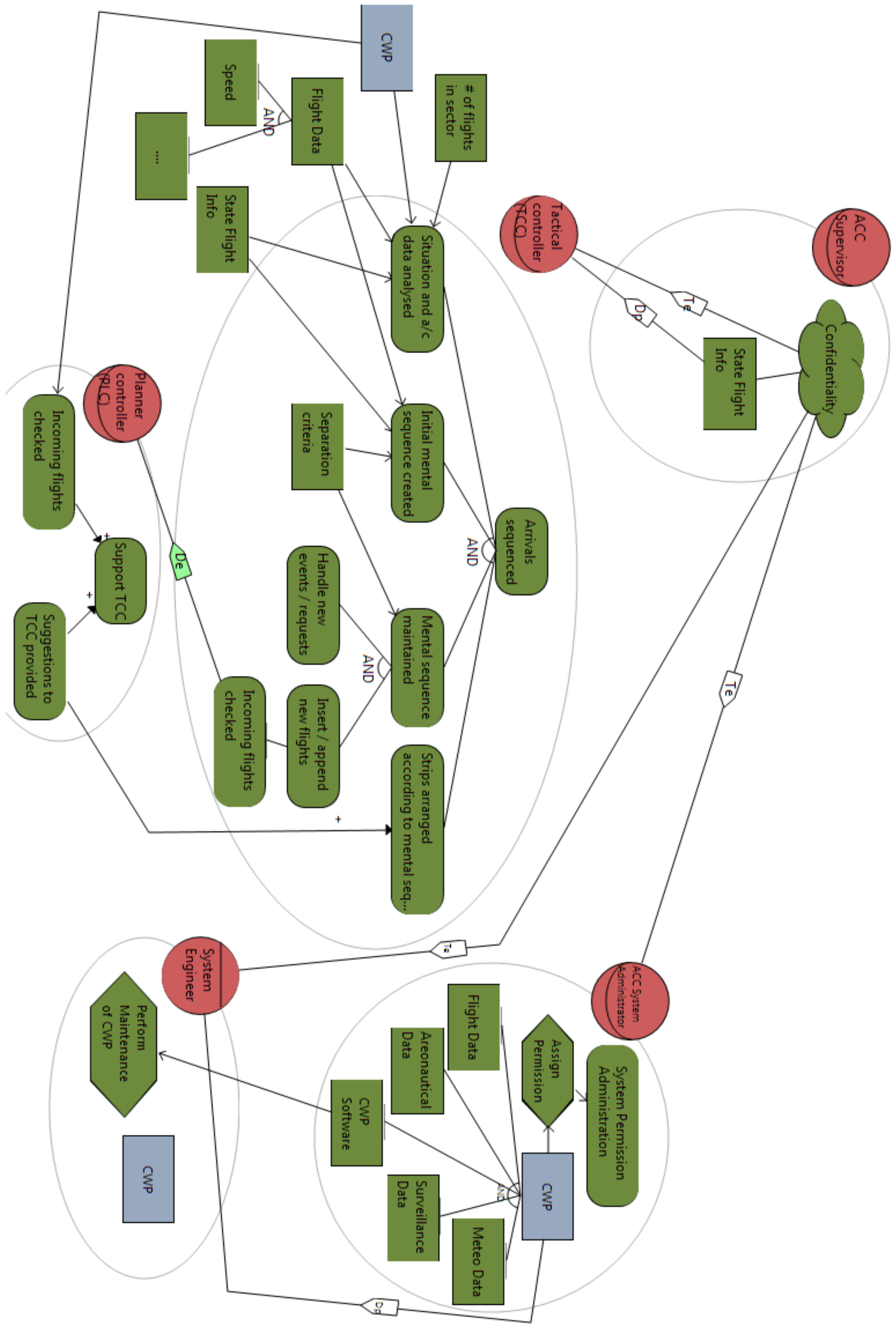
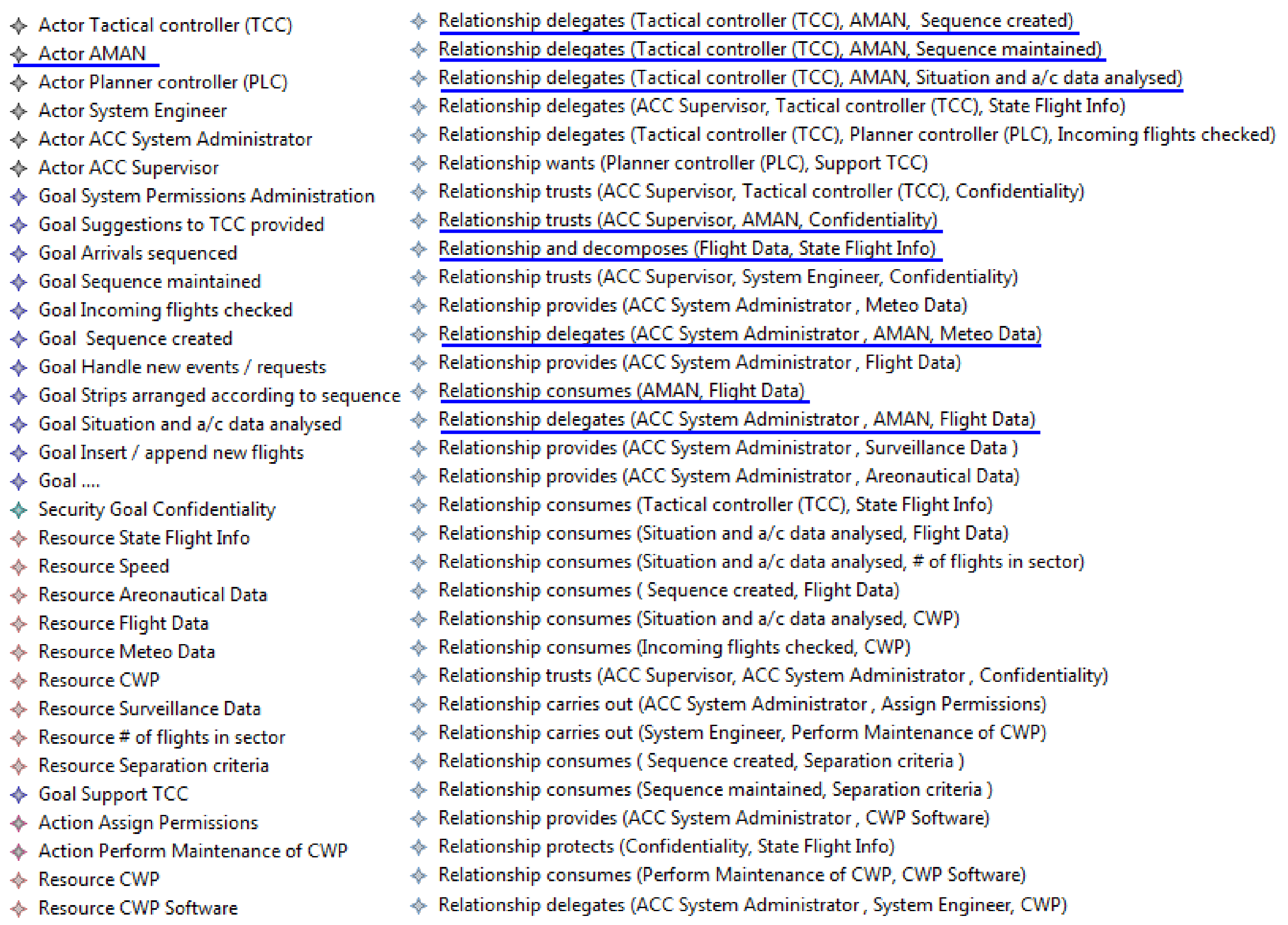


Figure 2. Evolution pre-state on Si* diagram

Then the change described in Section 2.1 occurs, and following editing actions are performed:

- The new Actor "AMAN" is created in the SeCMER model. The respective Actor immediately appears in the Si* diagram.
- On the other hand, in the Si* diagram a composition relationship is established to express that State Flight Info is now part of Flight Data. This makes some other edges unnecessary that represented the fact that State Flight Info is used to fulfill some goals (which is now implied by the containment in Flight Data); these connections are then deleted. The new or deleted relationships are immediately propagated to the abstract SeCMER model.
- Likewise, resource delegations and trust relationships targeting AMAN are created in the Si* diagram; these new connections will all immediately appear in the SeCMER model as well. The most important one expresses that AMAN is trusted with the "Confidentiality" Security Goal.

See Figure 3 and Figure 4 for the post-state of the model.



- Actor Tactical controller (TCC)
- Actor AMAN
- Actor Planner controller (PLC)
- Actor System Engineer
- Actor ACC System Administrator
- Actor ACC Supervisor
- Goal System Permissions Administration
- Goal Suggestions to TCC provided
- Goal Arrivals sequenced
- Goal Sequence maintained
- Goal Incoming flights checked
- Goal Sequence created
- Goal Handle new events / requests
- Goal Strips arranged according to sequence
- Goal Situation and a/c data analysed
- Goal Insert / append new flights
- Goal ....
- Security Goal Confidentiality
- Resource State Flight Info
- Resource Speed
- Resource Areonautical Data
- Resource Flight Data
- Resource Meteo Data
- Resource CWP
- Resource Surveillance Data
- Resource # of flights in sector
- Resource Separation criteria
- Goal Support TCC
- Action Assign Permissions
- Action Perform Maintenance of CWP
- Resource CWP
- Resource CWP Software

- Relationship delegates (Tactical controller (TCC), AMAN, Sequence created)
- Relationship delegates (Tactical controller (TCC), AMAN, Sequence maintained)
- Relationship delegates (Tactical controller (TCC), AMAN, Situation and a/c data analysed)
- Relationship delegates (ACC Supervisor, Tactical controller (TCC), State Flight Info)
- Relationship delegates (Tactical controller (TCC), Planner controller (PLC), Incoming flights checked)
- Relationship wants (Planner controller (PLC), Support TCC)
- Relationship trusts (ACC Supervisor, Tactical controller (TCC), Confidentiality)
- Relationship trusts (ACC Supervisor, AMAN, Confidentiality)
- Relationship and decomposes (Flight Data, State Flight Info)
- Relationship trusts (ACC Supervisor, System Engineer, Confidentiality)
- Relationship provides (ACC System Administrator , Meteo Data)
- Relationship delegates (ACC System Administrator , AMAN, Meteo Data)
- Relationship provides (ACC System Administrator , Flight Data)
- Relationship consumes (AMAN, Flight Data)
- Relationship delegates (ACC System Administrator , AMAN, Flight Data)
- Relationship provides (ACC System Administrator , Surveillance Data)
- Relationship provides (ACC System Administrator , Areonautical Data)
- Relationship consumes (Tactical controller (TCC), State Flight Info)
- Relationship consumes (Situation and a/c data analysed, Flight Data)
- Relationship consumes (Situation and a/c data analysed, # of flights in sector)
- Relationship consumes (Sequence created, Flight Data)
- Relationship consumes (Situation and a/c data analysed, CWP)
- Relationship consumes (Incoming flights checked, CWP)
- Relationship trusts (ACC Supervisor, ACC System Administrator , Confidentiality)
- Relationship carries out (ACC System Administrator , Assign Permissions)
- Relationship carries out (System Engineer, Perform Maintenance of CWP)
- Relationship consumes (Sequence created, Separation criteria)
- Relationship consumes (Sequence maintained, Separation criteria)
- Relationship provides (ACC System Administrator , CWP Software)
- Relationship protects (Confidentiality, State Flight Info)
- Relationship consumes (Perform Maintenance of CWP, CWP Software)
- Relationship delegates (ACC System Administrator , System Engineer, CWP)

Figure 3. Post-state in abstract SeCMER model (extract, changes marked)

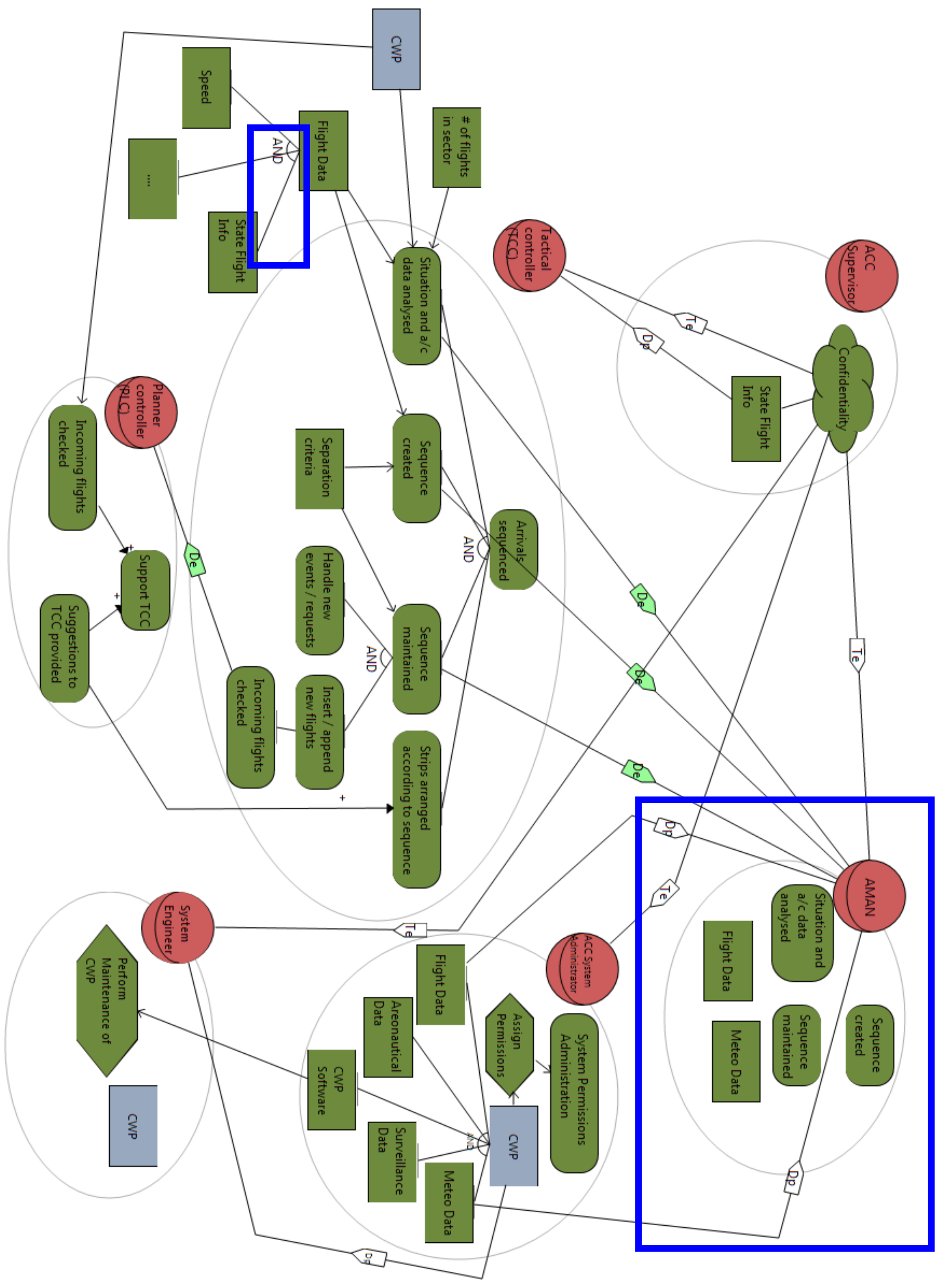


Figure 4. Evolution post-state on Si* diagram (changes marked)

2.4 Detection of a violation of least privilege principle and automatic corrective action based on evolution rules

Some security properties can be effectively evaluated or approximated based on the requirements model in a completely automated fashion. It is also possible to suggest default solutions. SeCMER includes the language of Evolution Rules, providing a declarative guard that can detect undesired situations and the possibility to include actions.

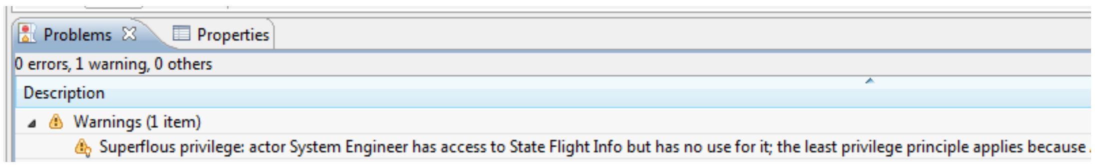
The following is a demonstration of on-the-fly evaluation of security properties and offering automated corrections using Evolution Rules.

Actor ACC Supervisor provides State Flight Info and wants to have its confidentiality preserved.

In the post-state, the asset forms a part of the aggregate Flight Data, which is accessible to various Control Room actors (e.g. via CWP), eventually including the System Engineer. Although System Engineer is trusted by the ACC Supervisor with the confidentiality security goal, there is no actual need for the former to have access to State Flight Info (as CWP maintenance only requires access to CWP Software, not the whole CWP containing valuable data assets). The tool marks this as a security violation of the least privilege principle (see Figure 5).

Automatic reactions (quick fixes) are also provided for this security pattern. Candidate solutions (see Figure 6) are automatically offered for each of the violations. In this case, the System Engineer only receives State Flight Info as part of Flight Data, so the only automatically offered solution is to add an action that expresses a procedure for which the System Engineer actually needs State Flight Info.

However rich the set of pre-defined quick fixes may be, there is always a possibility that the user chooses a solution that was not offered automatically. (See D.3.3 for research into generating potential solutions that were not defined a priori, and inductively learning the choices of the user.) In the example scenario, an alternative solution is to make sure that the System Engineer only receives those resources that it actually needs to perform its operational duties. In the current example, it may receive access to CWP Software, while not having access to the whole CWP, which would in turn provide access to the aggregate Flight Data (and therefore to State Flight Info).


Problems | Properties

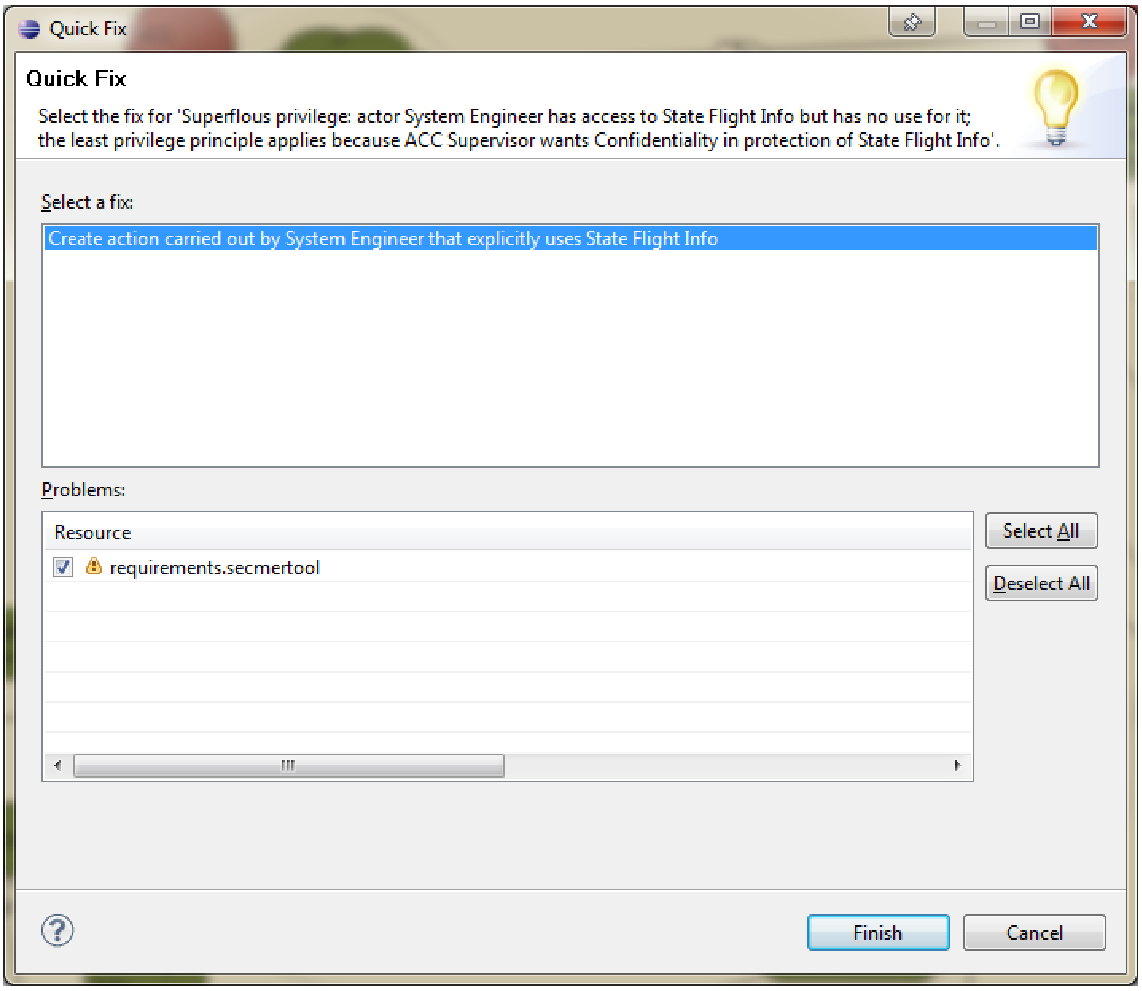
0 errors, 1 warning, 0 others

Description

Warnings (1 item)

Superflous privilege: actor System Engineer has access to State Flight Info but has no use for it; the least privilege principle applies because

Figure 5. Detected security issues


Quick Fix

Quick Fix

Select the fix for 'Superflous privilege: actor System Engineer has access to State Flight Info but has no use for it; the least privilege principle applies because ACC Supervisor wants Confidentiality in protection of State Flight Info'.

Select a fix:

Create action carried out by System Engineer that explicitly uses State Flight Info

Problems:

Resource
✓ requirements.secmertool

Figure 6. Automatic solutions suggested by evolution rules

2.5 Argumentation for the information access property

The scenario fragment we are going to consider is access of State Flight Info by the System Engineer via the enclosing Flight Data aggregate object, focusing on how to enforce confidentiality policies on State Flight Info. Arguments will be conducted for the security goal of preventing leaks of State Flight Info.

- Based on the requirements model, the facts that can be used for argumentation are initialized.
- Argumentation experts build the argument to support the claim that the system is secure. Counter-arguments (rebuttals) can be voiced, as well as mitigations for rebuttals, ad infinitum.
- Assuming that an initial argument existed in the pre-state, the effects of changes can also be represented as Round #2 rebuttals to facts. This means expanding the argument model with new arguments (marked as Round #2) that rebut or mitigate Round #1 arguments and facts. This chain of reasoning can continue onwards to Round #3 etc., signifying either subsequent changes or increasing depth of analysis.
- A traceability feature can track two kinds of links from arguments to elements of the requirements model:
    - An argument can be marked to support goals. The point of maintaining traceability of supported goals is that reports of security violations are suppressed if the violated security goal is supported by an argument. In other words, automatic problem detection is overridable by manually verified arguments.
    - Arguments, especially facts, can be linked to elements (*ground facts*) in the requirements model that they trace back to. This addresses the challenge in detecting arguments that have potentially been invalidated by changes to the requirements model. The benefit is that these arguments can be revisited to reflect the evolution, while no costly revision process is required for unaffected arguments. This is the purpose of keeping the traceability information on ground facts, so that model changes involving the ground facts may trigger a notification that alerts the user about the possibility that the argument may have become invalid due to the change.

The resulting Argumentation model is visualized in Figure 7. The diagram says that the State Flight Data is claimed to be confidential before the change (Round #1), and the claim is warranted by be the facts the ATCOs are known to have access to the flight data (F1), they reside in the Control Room (F2), etc. This argument is rebutted in Round #2, in which another argument claims that the system is no longer secure because Maintenance Engineers have superfluous access to State Flight Data. The rebuttal argument could be mitigated in Round #3 by further arguments.

Arguments like this can help the security engineers discover relevant Actors missing from the requirements model. In this case, Service Engineers were not included in the requirement model, but they were relevant for the security of State Flight Info. A possible next step could be to include this Actor to the requirements model as well.

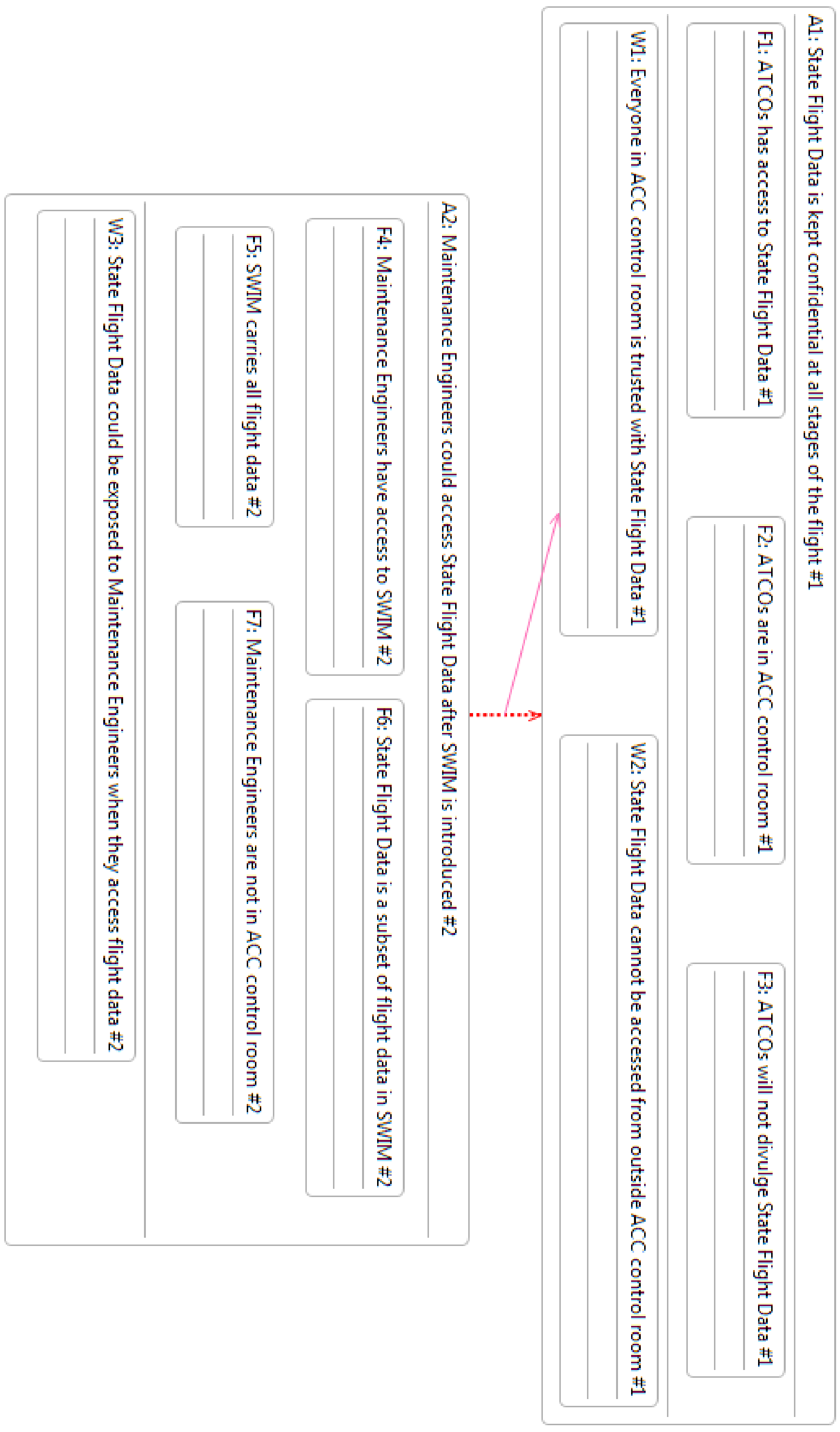

A1: State Flight Data is kept confidential at all stages of the flight #1
F1: ATCOs has access to State Flight Data #1
F2: ATCOs are in ACC control room #1
F3: ATCOs will not divulge State Flight Data #1
W1: Everyone in ACC control room is trusted with State Flight Data #1
W2: State Flight Data cannot be accessed from outside ACC control room #1
A2: Maintenance Engineers could access State Flight Data after SWIM is introduced #2
F4: Maintenance Engineers have access to SWIM #2
F6: State Flight Data is a subset of flight data in SWIM #2
F5: SWIM carries all flight data #2
F7: Maintenance Engineers are not in ACC control room #2
W3: State Flight Data could be exposed to Maintenance Engineers when they access flight data #2

Figure 7. A fragment of an argument model

3 Discussion

3.1 Related Work

There are many requirement engineering tools available but only some of them support specific capabilities for requirement change management. CASE Spec [6] makes it easy to generate traceability reports and perform impact analysis with built-in visual and tabular traceability tools. Dimensions RM [7] allows enterprises to effectively manage change in requirements during the project lifecycle. In particular, Dimensions RM facilitates the understanding of the impact of requirement changes and the creation of reports on requirements definition, baselines, change impact, and traceability. IBM Rational DOORS [8] has powerful capabilities for capturing, linking, analyzing and managing changes to requirements and their traceability. IBM Rational RequisitePro [9] is a requirements management tool that incorporates a powerful database infrastructure to facilitate requirements organization, integration, traceability and analysis. Moreover, it provides detailed traceability views that display parent/child relationships and shows requirements that may be affected by upstream or downstream changes. MKS Integrity 2009 [10] provides reuse and requirements change management capabilities coupled with meaningful (and traceable) relationships to downstream code and testing assets, which ensure communication of change, conformance to requirements and compliance with applicable governance or regulations.

Reqtify [11] is an interactive requirement traceability and impact analysis tool which can trace requirement from system, program and project levels to the entire levels of software or hardware component development lifecycle.

Compared with the above tools, SeCMER provides support to the requirement engineer for handling security related changes. The tool supports automatic detection of requirement changes that lead to violation of security properties using change-driven transformations and suggests possible corrective actions. The tool also supports argumentation analysis to check security properties are preserved by evolution and to identify new security properties that should be taken into account.

3.2 Known Technical Limitations

Unfortunately, there are a number of minor technical problems that have not yet been solved. None of these are blocker issues, and they do not interfere with the tool fulfilling its role as a prototype of several concepts and techniques developed in WP3.

Since to the composite model consists of multiple aspect models stored in separate files, the whole bundle of files makes up a single SeCMER model, and must be handled together as a unit. In particular, saving a copy of the model cannot be achieved by the “Save As…” command; it instead requires copying the entire folder of files together.

Due to the way the syntax grammar of the *.ontology* textual file format is currently defined, it does not support model elements with names containing spaces. An export to the .ontology format will only be successful if all SeCMER model elements have single-word names.

Since the SeCMER tool itself is an integration of several separate tools, undesirable interference between the various components may be observable in some circumstances; while great effort was exerted in this area, some of the sources of conflicts have not been isolated and resolved up to now. These issues usually manifest themselves as harmless messages in the Eclipse Log. In some cases they may emerge as user-facing error dialogs, but they are not known to actually impede work.

As an additional glitch caused by conflicts between the integrated components, the “dirty” (i.e. containing saveable changes) status of SeCMER editors may be incorrect in some circumstances. If the editor fails to show the dirty asterisk (*) in its name and does not let the user save the model, one can reinstate the “dirty” state e.g. by inserting a new model element and then deleting it via the "SeCMER" tab.

3.3 Foreseen Desirable Improvements

The prototype tool could potentially be improved through extension by other sets of security patterns to automate the detection and handling of security violations in a wider range of application scenarios. We plan also to realize a tighter integration with additional modeling formalisms (Problem Frames, CORAS) and industrial tools e.g DOORS-TREK.

The participating experts of the ATM workshop expressed an interest in a feature that can give an estimate of the level of security based on the requirements model, even when there are no violations conforming to the security patterns. Such a feature cannot be provided with the current set of WP3 techniques and would require further research before it could be integrated into the tool.

3.4 Conclusions

This document has presented SeCMER, a tool for managing evolving requirements. As shown by the ATM-based illustrative scenario, the tool supports visual modeling of security requirements. Additionally, argument models can be constructed manually to investigate the satisfaction of security properties; the tool detects invalidated arguments if the requirements model evolves. Finally, the tool performs continuous and automatic pattern-based violation detection of security properties, with optional “quick fix” corrective actions. Thus the tool integrates various ways of approaching security on the level of early requirements, to coherently support a workflow conforming to the SeCMER methodology (see D.3.2).

References

[1] The Eclipse Project: *Eclipse Modeling Framework*. http://eclipse.org/emf

[2] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, "Incremental Evaluation of Model Queries over EMF Models", *Model Driven Engineering Languages and Systems, 13th International Conference, MODELS'10*: Springer, 10/2010.

[3] F. Massacci, J. Mylopoulos, and N. Zannone, *Computer-aided Support for Secure Tropos.* Automated Software Engineering. **14**(3): p. 341-364, 2007.

[4] T. Tun, Y. Yu, R. Laney, and B. Nuseibeh, "Early Identification of Problem Interactions: A Tool-Supported Approach," in *Requirements Engineering: Foundation for Software Quality*, vol. 5512, Springer Berlin / Heidelberg, pp. 74-88, 2009.

[5] T. Tun, Y. Yu, C. Haley, B. Nuseibeh, "Model-Based Argument Analysis for Evolving Security Requirements," Secure System Integration and Reliability Improvement, pp. 88-97, 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement, 2010.

[6] CASE Spec, URL: http://computing-research.open.ac.uk/trac/openre

[7] Dimensions RM, URL: http://www.serena.com/products/rm/index.html

[8] IBM Rational DOORS, URL: http://www-01.ibm.com/software/awdtools/doors/

[9] IBM Requisite Pro, URL: http://www-01.ibm.com/software/awdtools/reqpro/

[10] IMKS Integrity 2009, URL: http://www.mks.com/

[11] Geensoft Reqtify, URL: http://www.geensoft.com/en/article/reqtify

Appendix A. Tool Realization

In this Section we give an overview of the design choices behind the actual prototype tool and the integration of the academic tools.

A.1. Architectural Overview

A prototype tool is realized as a set of Eclipse plug-ins written in Java (partly generated), and models are represented in the Eclipse Modeling Framework (EMF [1]). The components of the tool currently fall into these categories:

- Eclipse plug-ins of OpenPF (requirements engineering tool by OU [4]), including (a) the implementation of the SeCMER conceptual model, (b) the argumentation model and tools, as well as (c) the modeling tools for Problem Frames (only limited synchronization is supported with the other formalisms as of now);
- Si* (requirements engineering tool [3] by UNITN);
- traceability models to represent the relationship between corresponding model elements in different languages, e.g. the SeCMER conceptual model and Si*;
- run-time platform components of EMF-INCQUERY (incremental EMF model query engine by BME [2]) for change-driven transformations;
- model query plug-ins automatically generated from transformation specification and Evolution Rules by the development-time tools of EMF-INCQUERY and VIATRA2 (model transformation framework by BME);
- integration code developed solely for this tool, including User Interface commands and the Java definition of the action parts of Evolution Rules.

The relationship of the most important model management components are depicted on , focusing on the Si* and SeCMER models in particular, as well as the traceability model established between them. User Interface components are omitted from this diagram. See also Section 1 for a higher-level overview of what components and modeling formalisms are involved.

All the involved EMF models are accessed through a common EMF ResourceSet and edited solely through the corresponding TransactionalEditingDomain (from the EMF Transaction API). Consequently, all modifications are wrapped into EMF Transactions, including those carried out by manual editing through the User Interface (e.g. the Si* diagram editor or the generic EMF tree editor) as well as changes performed by automated mechanisms such as model transformation. As one of the benefits, concurrent modifications are serialized and therefore conflict-free. Furthermore, the commit process of the transactions provides an opportunity for triggering change-driven actions.

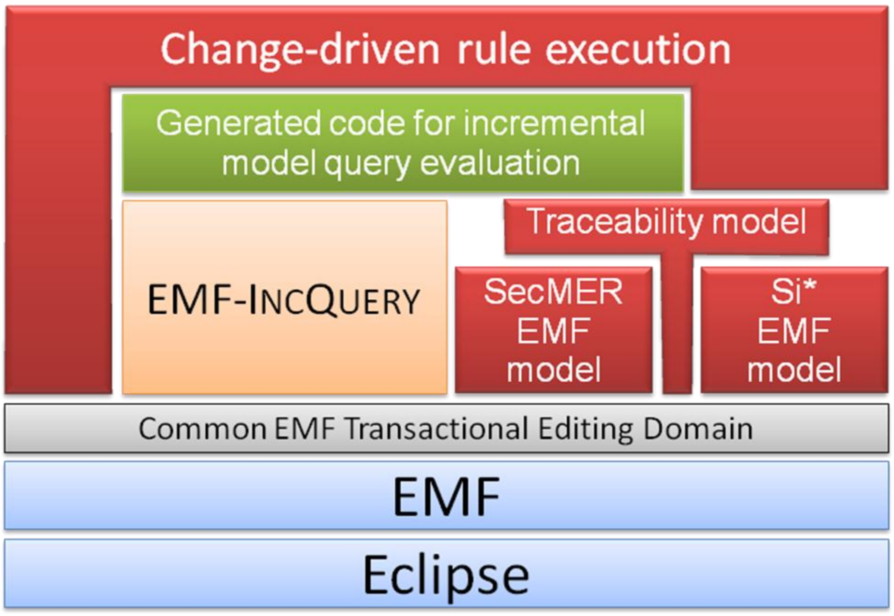

Change-driven rule execution
Generated code for incremental model query evaluation
EMF-INCQUERY
Traceability model
SecMER EMF model
Si* EMF model
Common EMF Transactional Editing Domain
EMF
Eclipse

Figure 8. Architectural overview of model management components

A.2. Trigger mechanism

The incremental query mechanism provided by EMF-INCQUERY plays a key role in the functionality of the tool. Incremental query evaluation code is generated automatically at development time by EMF-INCQUERY, from a graph pattern-based declarative description of EMF model queries. Through this incremental evaluation functionality, Evolution Rules can be efficiently triggered by changes captured as graph patterns. The implementation currently supports detecting the presence, appearance and disappearance, but not the entire Graph Change Patterns formalism (see D.3.2).

The core triggering plug-in offers an Eclipse extension point for defining change-driven rules. Multiple constituent plug-ins contribute extensions to active their respective set of rules. The graph pattern-based declarative event/condition feature of the rules is evaluated by the incremental graph pattern matcher plug-ins automatically generated from the declarative description by EMF-INCQUERY. At the commit phase of each transaction, the rules that are found to be triggered will be executed to provide their reactions to the preceding changes. These reactions are implemented by arbitrary Java code, and they are allowed to modify the model as well (wrapped in nested transactions) and could therefore be reacted upon.

Currently, there are three groups of change-driven rules contributed to the extension point:

- transformation rules that realize the on-the-fly synchronization between multiple modeling formalisms,

- security-specific evolution rules that detect the appearance of undesired security patterns, provide alerts on these problems and optionally offer candidate solutions.
- rules for marking arguments as invalid when changes are inflicted on their ground facts.

A.3. Transformation specification

The tool maintains a synchronizing transformation between Si* and the SeCMER model. The challenge is to provide bi-directional synchronization with changes propagated on the fly. Naturally the two languages have different expressive power, therefore

(a) some concepts are not mapped from one formalism to the other or vice versa,

(b) some model elements may be mapped into multiple (even an unbounded amount of) corresponding model elements in the other formalism, and finally

(c) it is possible that a single model element has multiple possible translations (due to the source formalism being more abstract); one of them is created as a default choice, but the other options are also accepted.

The following mappings define the transformation:

- There is a many-to-one correspondence between Si* Actors of the same name and SeCMER Actors.
- There is a many-to-one correspondence between Si* Resources with the same name and SeCMER Resources that are provided (thus eligible to be represented in Si*).
- There is a one-to-one correspondence between the original copy of a Si* Resource, owned by an Actor and not received through delegation, and the SeCMER Provides Relationship from the corresponding Actor to the corresponding Resource.
- Additional copies of a Si* Resource, owned by an Actor and received by delegation but not delegated further, are mapped into SeCMER Consumes Relationships from the corresponding Actor to the corresponding Resource.
- There is a many-to-one correspondence between Si* Tasks with the same name and SeCMER Actions that are carried out (thus eligible to be represented in Si*).
- There is a one-to-one correspondence between a copy of a Si* Task owned by an Actor and not delegated further, and the SeCMER Carries Out Relationship between the corresponding Actor and Action.
- There is a many-to-one correspondence between Si* Softgoals having the same name and SeCMER Security Goals that are wanted (thus eligible to be represented in Si*).
- There is a many-to-one correspondence between Si* Goals of the same name, and SeCMER Goals that are wanted (thus eligible to be represented in Si*).and

are *not* Security Goals. A newly created Si* Goal is mapped *by default* into a SeCMER Goal of type Goal proper (not any of its subtypes).

- There is a one-to-one correspondence between the orginial copy of a Si* Goal or Softgoal owned by an Actor and not received by delegation, and the SeCMER Wants Relationship between the corresponding Actor and Goal.
- There is a one-to-one correspondence between Si* ‘AND’ Compositions and SeCMER And Decompositions between Actions or Goals, if both endpoints are mapped.
- There is a one-to-one correspondence between Si* ‘OR’ Compositions and SeCMER Or Decompositions between Actions or Goals, if both endpoints are mapped.
- There is a one-to-one correspondence between a Si* MeansEnd Relation from a Task to a Goal or Softgoal and the SeCMER Fulfils Relationship between the corresponding Action and Goal, if both endpoints are mapped.
- There is a one-to-one correspondence between a Si* MeansEnd Relation from a Resource to a Task or Goal and the SeCMER Consumes Relationship between the corresponding Action / Requirement and Resource, if both endpoints are mapped.
- There is a one-to-one correspondence between a Si* MeansEnd Relation from a Task or Goal to a Resource and the SeCMER Produces Relationship between the corresponding Action / Requirement and Resource , if both endpoints are mapped.
- There is a one-to-one correspondence between a Si* ‘Custom’ Relation from a Softgoal to a Resource / Task and the SeCMER Protects Relationship between the corresponding Security Goal and asset, if both endpoints are mapped.
- There is a one-to-one correspondence between a Si* Delegation of Permission Relation, pointing from a Resource owned by an Actor to a second Actor, and the SeCMER Delegates Relationship with the first Actor as source, the second Actor as target and the Resource as dependum, if all three endpoints are mapped.
- There is a one-to-one correspondence between a Si* Delegation of Execution Relation, pointing from a Task or Goal or Softgoal owned by an Actor to a second Actor, and the SeCMER Delegates Relationship with the first Actor as source, the second Actor as target and the Action or Goal as dependum, if all three endpoints are mapped.
- There is a one-to-one correspondence between a Si* Trust of Permission Relation, pointing from a Resource owned by an Actor to a second Actor, and the SeCMER Trusts Relationship with the first Actor as source, the second Actor as target and the Resource as dependum, if all three endpoints are mapped.
- There is a one-to-one correspondence between a Si* Trust of Execution Relation, pointing from a Task or Goal or Softgoal owned by an Actor to a second Actor, and the SeCMER Trusts Relationship with the first Actor as

source, the second Actor as target and the Action or Goal as dependum, if all three endpoints are mapped.

Appendix B. User Manual

B.1. Installation

There are two possible ways to obtain the SeCMER tool. The first is a simple installation process aimed at security engineers that only want to use the tool; the second, more elaborate description is for developers who want to study or modify the project sources.

B.1.1. Recommended Installation for End-Users

This distribution of the tool is targeted to run on a Windows machine with 32-bit Java. If you use a different OS or 64-bit Java, you can deploy an Eclipse instance with the necessary components yourself, based on the developer instructions in Section B.1.2.

Please download and extract the archive file at:

http://securechange.eu/sites/default/files/tools/secmer/deployed.7z

The folder `deployed/` contains a Win32 copy of Eclipse Modeling 3.6.0 with all components of the SeCMER tool. You can start the SeCMER tool by running `eclipse.exe`. When it asks for a workspace directory, it is recommended to select the folder `runtime-SecMER/` that can be obtained as described in Section B.1.3.

B.1.2. Installation of Development Environment

This type of installation is only necessary if you plan to contribute to the source code of the tool, or if you need to deploy an instance of the tool onto your preferred platform (such as 64-bit Java or Mac OS X).

Please download and extract the archive file at:

http://securechange.eu/sites/default/files/tools/secmer/demonstrator.7z

Most versions of the bundle contain a **Win32 copy of Eclipse Modeling 3.6.0** with all necessary prerequisite components for the development of the SeCMER tool. You can start the development Eclipse instance by running `eclipse-3.6/eclipse.exe`.

This Eclipse instance may be omitted for space considerations from some distributions of the tool. If this folder is missing, or you work in an other OS or use 64-bit Java, you can **set up an appropriate instance of Eclipse yourself**. Start by installing a distribution of Eclipse 3.6 (Helios) that is appropriate on your system. Please also install the feature `SWTBot for Eclipse` from its respective update site, as it is a required dependency. Additionally, the following two folders (distributed with the bundle) need to be copied into to the `dropins/` subdirectory of your Eclipse 3.6 installation: `hu.bme.mit.incquery.dropins/` (contains a snapshot of Viatra2 and EMF-INCQUERY components) and `open.pf.prerequisites.dropins/` (contains prerequisite dependencies of OpenPF). In the supplied `eclipse-3.6/` folder, these dropins are already copied into place, so you do not need to do anything if you use a Win32 environment.

When starting the development environment, the folder `dev-workspace/` (also contained in the archive bundle) should be selected as its **workspace**. The workspace consists of the following Working Sets of projects:

- The Working Set `OpenPF` consists of a checked out copy of an SVN tagged version of Open University's OpenPF.
- The Working Set `Si*`, on the other hand, contains a snapshot of UNITN's Sistar tool. Some very slight alterations have been applied to fit the tool into the framework (mostly related to sharing the EMF `TransactionalEditingDomain` between tools).
- The Working Set `CASE Tool` contains the plug-in projects that constitute the prototype SeCMER Tool responsible for the integration of the other tools. Its goals involve providing traceability, synchronizing transformations and other features (most notably incremental evaluation of security patterns).
- The Working Set `Releng` contains Eclipse Feature definitions that describe how to make the tool installable.

The development eclipse instance has the Java Development Tools, the Eclipse Plugin Development Environment and Viatra2 installed for the development of the SeCMER CASE Tool. The Java and Viatra2 perspectives are made available for this purpose.

You can start **debugging the prototype tool** by selecting the "`SecMER Demonstrator Case Tool`" launcher (should be default actually) from the dropdown menu of the Debug button (green bug) on the toolar. Switching to the Java perspective might be necessary for the Debug button to show up.

To **deploy a stand-alone instance of the SeCMER tool**, you should create a local Eclipse Update Site of the SeCMER Tool components (the folder `featuredeploy/` is provided for this purpose). Use File/Export > "Plug-in Development"/"Deployable features" to populate the update site folder from the current version of the code in the workspace. Select all features in the dialog box, and specify this `featuredeploy` folder as Destination/Directory.

You can then create a deployed instance of the tool (such as the one described in Section B.1.1) by preparing a separate instance of Eclipse similar to the development environment, including the two dropin folders indicated above; then adding this `featuredeploy/` folder as a local update site, and finally installing the SeCMER Case Tool components from it.

B.1.3. Example Artifacts

To obtain the example models, please download and extract the archive file at:

http://securechange.eu/sites/default/files/tools/secmer/runtime-SeCMER.7z

The folder `runtime-SecMER/` contains the Eclipse workspace of the tool prototype. The project `DemoCorner` holds all the example artifacts that correspond to the Feature Tour in Section 2. The file `DemoCorner/before/requirements.secmertool` is an example model for the tool containing the pre-state (before state) of the evolution, while `DemoCorner/after/requirements.secmertool` is a sample model containing the post-state (after state) of the evolution. The SeCMER tool uses EMF models with the `.secmertool` extension. These files references aspect models (SeCMER model, Si*, OpenPF Arguments, more to come) from the `aspect-models` folder.

B.2. User Interface

B.2.1. Viewing and editing models with the SeCMER Tool

To start using the tool, open any file with the extension `.secmertool`, such as the ones delivered as example artifacts (see Section B.1.3)

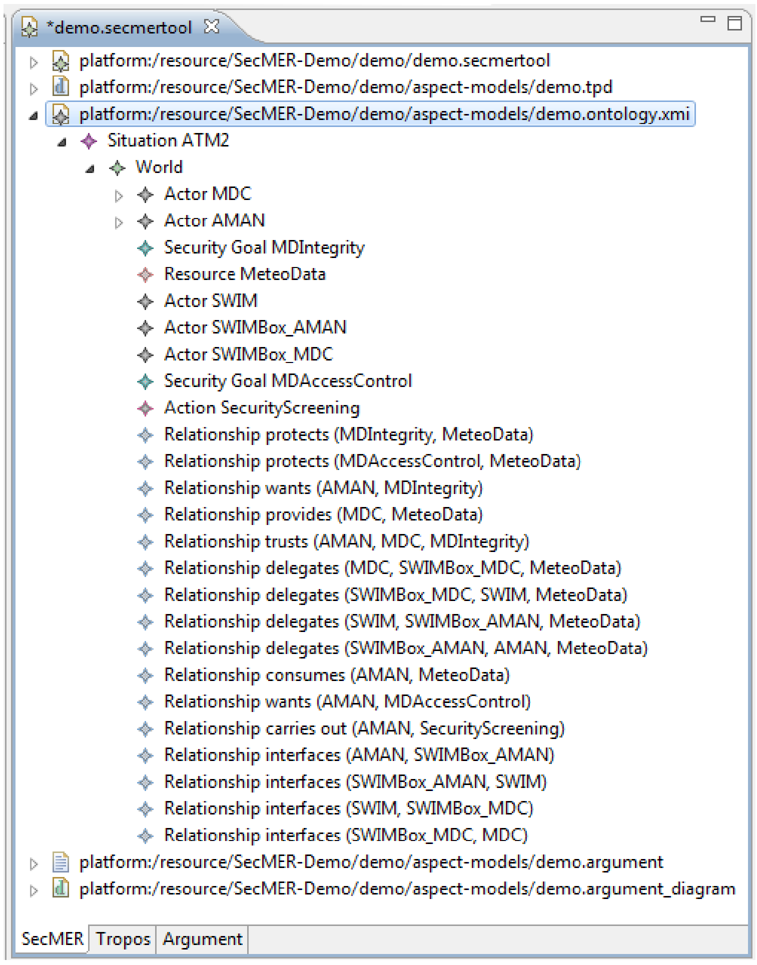
IMPORTANT NOTE: do NOT open directly any models in the `aspect-models/` subfolder. Use the functionality provided by the SeCMER editor and exporters.

In the **"SecMER" tab** of the tree editor of the `.secmertool` EMF resource, you can view the abstract representation of the contents of all associated models. **Most users can simply ignore this view.** For advanced users, here is the structure of the contents of this view:

- Within the `.secmertool` file proper (appearing under "`platform:/...`*`something`*`.../`*`somefile`*`.secmertool`"), the element "SecMER Integration Model" is responsible for gluing together the various other models, and contains traceability information.
- Under "`platform:/...`*`something`*`.../`*`somefile`*`.ontology.xmi`", you can find the SeCMER requirements model. What you see is actually the abstract EMF model converted from the textual representation of the requirements model.
- Under "`platform:/...`*`something`*`.../`*`somefile`*`.argument`", you can find the argumentation model associated with the SeCMER requirements model. What you see is actually the abstract EMF model transparently parsed from the textual representation of the argumentation model.
- Under "`platform:/...`*`something`*`.../`*`somefile`*`.tpd`", you can view two subtrees contained in the Si* file: the abstract Tropos Model, and the abstract structure of the graphical diagram elements.

The tree editor can be used in conjunction with the Eclipse Properties View to manipulated the abstract form of the SeCMER Requirements model.

For example, to create a “`Protects`” relationship between an existing Security Goal and an Asset, right-click the “`World`” element in the subtree of “`...ontology.xmi`”, issue `New Child > Relationship`, select the newly created element, and finally fill out its property sheet. This last step requires the Properties View (can be opened by right-clicking the element and issuing `Show Properties View`); `Dependum` should be left empty, `Source` should be set to the Security Goal and `Target` should be set to the asset that it protects; finally Type should be set to “`protects`”.



*demo.secmertool

- platform:/resource/SecMER-Demo/demo/demo.secmertool
- platform:/resource/SecMER-Demo/demo/aspect-models/demo.tpd
- platform:/resource/SecMER-Demo/demo/aspect-models/demo.ontology.xmi
  - Situation ATM2
    - World
      - Actor MDC
      - Actor AMAN
      - Security Goal MDIntegrity
      - Resource MeteoData
      - Actor SWIM
      - Actor SWIMBox_AMAN
      - Actor SWIMBox_MDC
      - Security Goal MDAccessControl
      - Action SecurityScreening
      - Relationship protects (MDIntegrity, MeteoData)
      - Relationship protects (MDAccessControl, MeteoData)
      - Relationship wants (AMAN, MDIntegrity)
      - Relationship provides (MDC, MeteoData)
      - Relationship trusts (AMAN, MDC, MDIntegrity)
      - Relationship delegates (MDC, SWIMBox_MDC, MeteoData)
      - Relationship delegates (SWIMBox_MDC, SWIM, MeteoData)
      - Relationship delegates (SWIM, SWIMBox_AMAN, MeteoData)
      - Relationship delegates (SWIMBox_AMAN, AMAN, MeteoData)
      - Relationship consumes (AMAN, MeteoData)
      - Relationship wants (AMAN, MDAccessControl)
      - Relationship carries out (AMAN, SecurityScreening)
      - Relationship interfaces (AMAN, SWIMBox_AMAN)
      - Relationship interfaces (SWIMBox_AMAN, SWIM)
      - Relationship interfaces (SWIM, SWIMBox_MDC)
      - Relationship interfaces (SWIMBox_MDC, MDC)
- platform:/resource/SecMER-Demo/demo/aspect-models/demo.argument
- platform:/resource/SecMER-Demo/demo/aspect-models/demo.argument_diagram

SecMER | Tropos | Argument

Figure 9. Tree editor of the abstract model

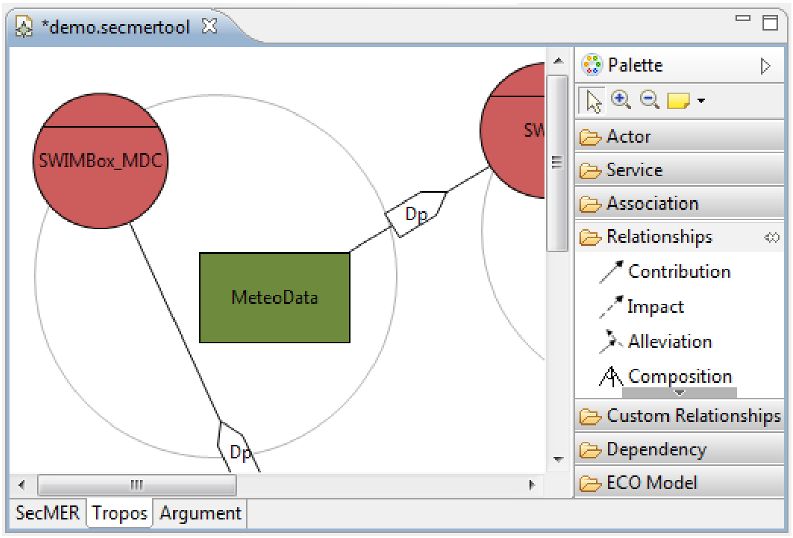


Figure 10. Si* diagram on the "Tropos" tab

The editor has two more tabs. The **"Tropos" tab** contains the Si* Tropos diagram editor to show the graphical representation of the Si* aspect. This tab can be used as a regular Si* editor. Users typically perform most of the requirement modeling using this view, since modifications are automatically propagated between the abstract SeCMER requirement model seen in the tree editor, and the Si* requirements model.

There are certain rare types of model elements, though, that are not represented in the Si* syntax; none of them is particularly important for security analysis or the demo scenario.

Finally, the **"Argument" tab** shows the argument diagram for the Argumentation model associated with the requirements model.

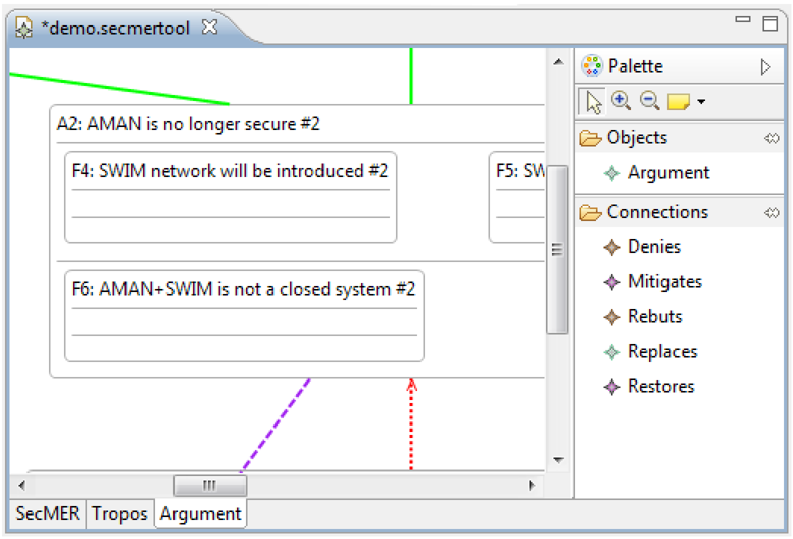


Figure 11. Argument diagram on the “Argument” tab

B.2.2. Creating new models with the SeCMER Tool

New SeCMER models should be created using one of the “New SeCMER model” wizards. Shortcuts for these wizards under the `File > New` menu item and under the `New` toolbar button have been configured in the SeCMER perspective; and the wizards are always available at `File > New > Other... > SeCMER Tool` regardless of the selected perspective.

To create a `.secmertool` model from scratch, invoke the `New SeCMER model from scratch` wizard. Select the location (container folder) of the model, and then specify a file name. A new and empty SeCMER integrated model will be created, along with all the aspect models in a separate folder.

To create a `.secmertool` model from a pre-existing Tropos / Si* model, invoke the `New SeCMER model from Tropos / Si* model` wizard. Before providing the target container and file name, an input Tropos model has to be selected. The resulting .secmertool model will contain a copy of the specified Tropos model as one of its aspects.

Similarly, the `New SeCMER model from .ontology file` wizard creates a `.secmertool` model, whose SeCMER abstract model aspect will be initialized according to a SeCMER model in the `.ontology` textual syntax.

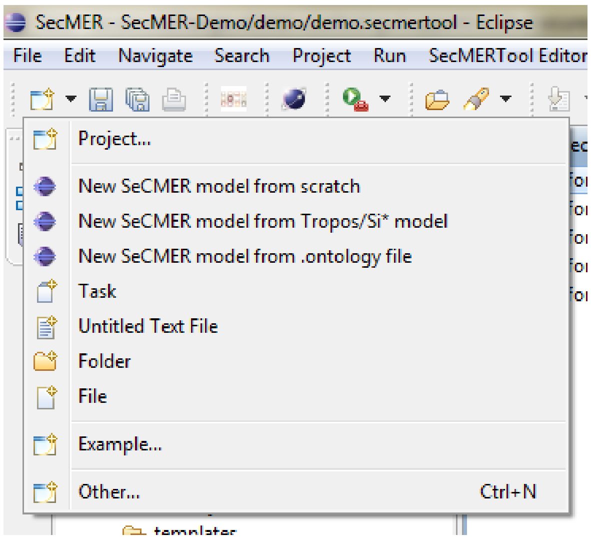

SecMER - SecMER-Demo/demo/demo.secmertool - Eclipse
File Edit Navigate Search Project Run SecMERTool Editor
Project...
New SeCMER model from scratch
New SeCMER model from Tropos/Si* model
New SeCMER model from .ontology file
Task
Untitled Text File
Folder
File
Example...
Other... Ctrl+N

Figure 12. New SeCMER Model wizards

B.2.3. Exporting models from the SeCMER Tool

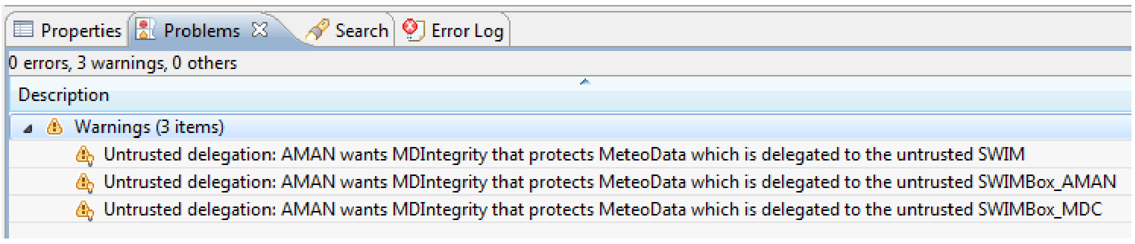
Aspect models should only be viewed and edited through the SeCMER editor; directly manipulating the Aspect models is not supported. However, they can be exported from the SeCMER model by right-clicking the `.secmertool` file in the `Project Explorer`, and issuing an export command from the "`SeCMER Tool`" submenu.

Invoking the `Export SeCMER model into .ontology file` exports the requirements model into a selected file with the `.ontology` textual syntax. Invoking `Export Tropos / Si* model` will export the Si* aspect into the selected `.tpd` file. Finally, Export Argument Model extracts the argument model in the `.argument` textual syntax.

B.2.4. Detecting and fixing security violations

Some security properties can be effectively evaluated or approximated based on the requirements model in a completely automated fashion. Each such detected security violation shows up as a Warning marker in the "`Problems`" View of the Eclipse Workbench (see Figure 13).

Default solutions or solution templates are also automatically suggested. Solutions for a particular problem can be discovered by right clicking the appropriate Warning marker in the "`Problems`" View, and then issuing "`Quick fix`". A dialog will appear listing possible ways to resolve the violation in question; they can be selected and executed automatically (see Figure 14).



0 errors, 3 warnings, 0 others

Description

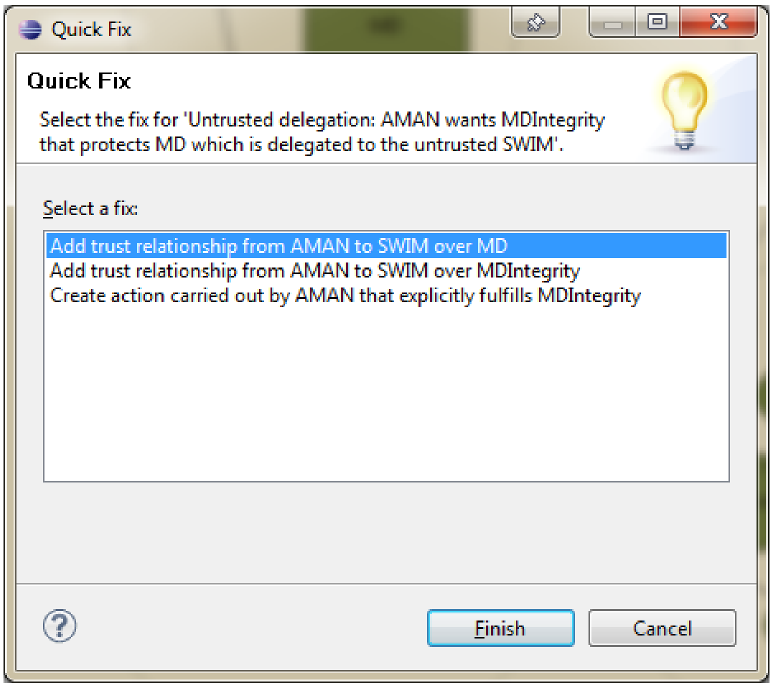
Warnings (3 items)

Untrusted delegation: AMAN wants MDIntegrity that protects MeteoData which is delegated to the untrusted SWIM

Untrusted delegation: AMAN wants MDIntegrity that protects MeteoData which is delegated to the untrusted SWIMBox_AMAN

Untrusted delegation: AMAN wants MDIntegrity that protects MeteoData which is delegated to the untrusted SWIMBox_MDC

Figure 13. Detected security issues



Quick Fix

Select the fix for 'Untrusted delegation: AMAN wants MDIntegrity that protects MD which is delegated to the untrusted SWIM'.

Select a fix:

Add trust relationship from AMAN to SWIM over MD
Add trust relationship from AMAN to SWIM over MDIntegrity
Create action carried out by AMAN that explicitly fulfills MDIntegrity

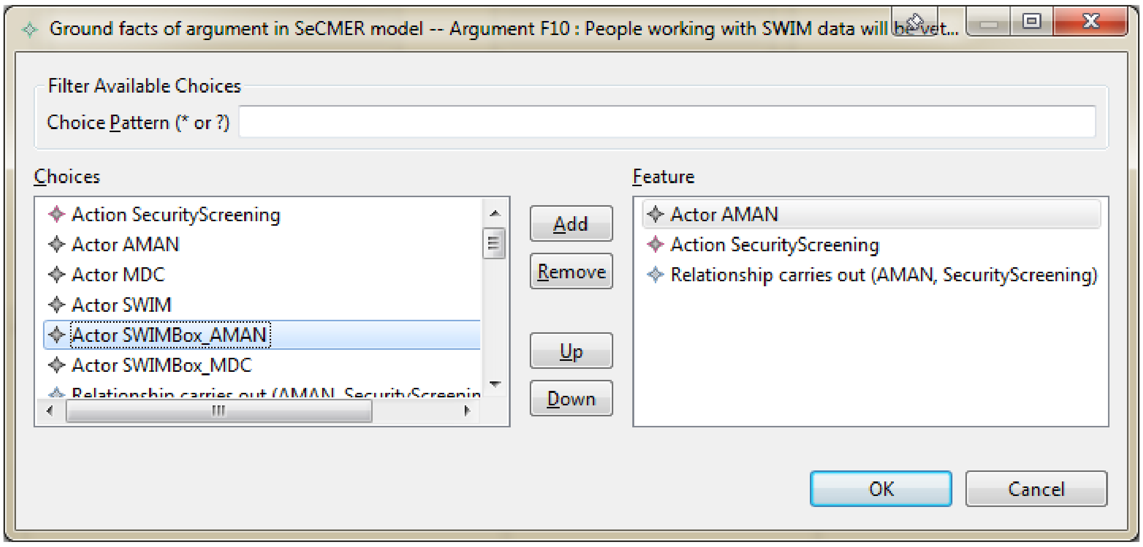
Figure 14. Automatically suggested solutions

B.2.5. Arguments to requirements traceability

SeCMER currently supports two kinds of traceability information that connect the Argument model and the Requirements model:

- The **ground facts** or *evidence* of an argument (typically an empty argument, i.e. fact) are Requirements model elements about which the argument states a proposition. Many arguments (typically the composite ones) don't have ground facts.
- Some top-level arguments may have **supported goals** in the requirements model. If the argument is valid, then the goals can be considered satisfied.

These traceability links can be established by right-clicking an argument box in the Argument Diagram, and issuing either `Ground facts in SeCMER model…` or `Supported goals in SeCMER model…` from the context menu. A dialog will appear, with two panes. The pane on the left shows requirements model elements / goals that can be potentially selected as ground facts / supported goals, while the pane on the right lists those that are currently selected as such. To add a ground fact / supported goal, select it from the list of choices (left pane), and push “Add”. To remove a previously chosen element, select it on the right pane, and hit “Remove” (see Figure 15).


Ground facts of argument in SeCMER model -- Argument F10 : People working with SWIM data will be vet...

Filter Available Choices
Choice Pattern (* or ?)

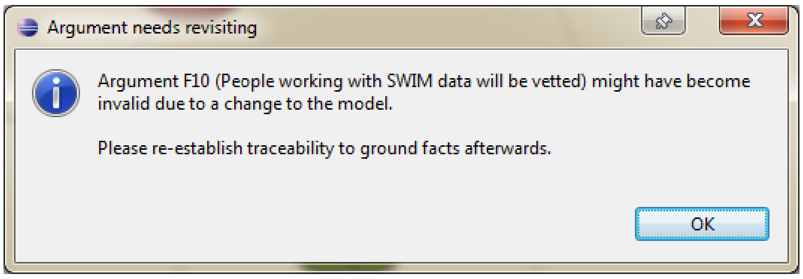
Choices
- Action SecurityScreening
- Actor AMAN
- Actor MDC
- Actor SWIM
- Actor SWIMBox_AMAN
- Actor SWIMBox_MDC
- Relationship carries out (AMAN, SecurityScreenin

Feature
- Actor AMAN
- Action SecurityScreening
- Relationship carries out (AMAN, SecurityScreening)

Figure 15. Selecting ground facts for Argument F10

The point of maintaining traceability of supported goals is that reports of security violations are suppressed from the Problems View if the violated security goal is supported by an argument. In other words, automatic problem detection is overridable by manually verified arguments.

While argumentation is a powerful framework for early-stage analysis of security properties based on the requirements model, by default it considers a single state of the model. The challenge is in detecting arguments that have potentially been invalidated by changes, and revisiting these arguments to reflect the evolution, while no costly revision process is required for unaffected arguments. This is the purpose of keeping the traceability information on ground facts, so that model changes involving the ground facts may trigger a notification that alerts the user about the possibility that the argument may have become invalid due to the change.


Argument needs revisiting

Argument F10 (People working with SWIM data will be vetted) might have become invalid due to a change to the model.

Please re-establish traceability to ground facts afterwards.

Figure 16. Detecting the invalidation of an argument

Appendix C. Report on initial industrial evaluation of the EMF-INCQUERY tool

In the context of SecureChange, Thales evaluated the applicability of the EMF-INCQUERY framework to its industrial context.

C.1. Industrial context

In order to build an architecture of a software intensive system, many stakeholders contribute to the description of the system architecture. Following a model-based engineering approach, the different stakeholders will use modelling tools to describe the architecture and analysis tools to evaluate some properties of the architecture.

Thales has defined a model-based architecture engineering approach for software intensive systems, the ARCADIA method. It defines a model organization of five abstraction levels (viewpoints) for mainstream engineering and a set of other viewpoints for speciality engineering, depending typically on non-functional constraints applied on the system to be engineered. The views conforming to these viewpoints are used by different stakeholders during the system definition process. Therefore, techniques and tools to manage the consistency of an information bulk made of several views on a system are necessary. The ARCADIA method adopts a viewpoint-based architectural description such as described in the conceptual foundations of ISO/IEC 42010, Systems and Software Engineering - Architecture Description.

This ongoing standard attempts to specify the manner in which architecture descriptions of systems are expressed. This standard provides key concepts and their relationships to document an architecture. Its key concepts are Architecture Framework, Architecture Description, Viewpoint, View and Correspondence rule. An architecture description aggregates several Architecture Views. A view addresses one or more system concerns that are relevant to some of the system's stakeholders. A view aggregates one or more Architecture Models. Each view is defined following the conventions of an Architecture Viewpoint. The viewpoint defines the Model Kinds used for that view to represent the architecture addresses stakeholders' concerns.

As stated in this standard, in architecture descriptions, one consequence of employing multiple views is the need to express and maintain the consistency between these views. The standard introduces the Correspondence Rule concept that states a constraint that must be enforced on a correspondence to express relation between architecture description elements (Views, Architectural Model, etc.). Correspondences can be used to express consistency, traceability, composition, refinement and model transformation, or dependencies of any type spanning more than a single model kind.

Considering this industrial context, it can be considered that there are 3 major types of model consistency to manage:

- The first one aims at ensuring that a model conforms to its meta-model, i.e. that it addresses the well-formedness of the model. Since the modelling environment is DSL based (i.e. not profile based using a general purpose language), the well-formedness can be de facto ensured.

- The second one aims at ensuring that a model conforms to a coherent set of engineering rules; i.e. that the engineer conforms to a defined engineering method; in order to capitalize and reuse standard and domain specific engineering best practices.
- The third one aims at ensuring information consistency between distributed engineering environments, i.e. when there is not a unique centralized data reference. The main purpose here is to ensure coherency of all engineering activities across engineering domains, typically mainstream architecting and speciality engineering activities.

C.2. Objectives

Our experimentation focused on the second type of model coherency which consists in determining if a given configuration of set of views (models) are coherent with a set of consistency rules or not. This study consisted in detecting if the inconsistencies between views conformed to the engineering meta-model defined by Thales and composed of a set of 20 meta-models and about 400 meta-classes involved in the five viewpoints defined in ARCADIA. The purpose of this study was assessing the benefits of EMF-INCQUERY when compared to other environments: the traditional Java over EMF one, and the incremental Prolog-based Praxis. In terms of benefits, we studied the usability of the approach.

We validated the approach by translating a set of existing consistency rules initially implemented in Java over EMF into EMF-INCQUERY.

C.3. Environment

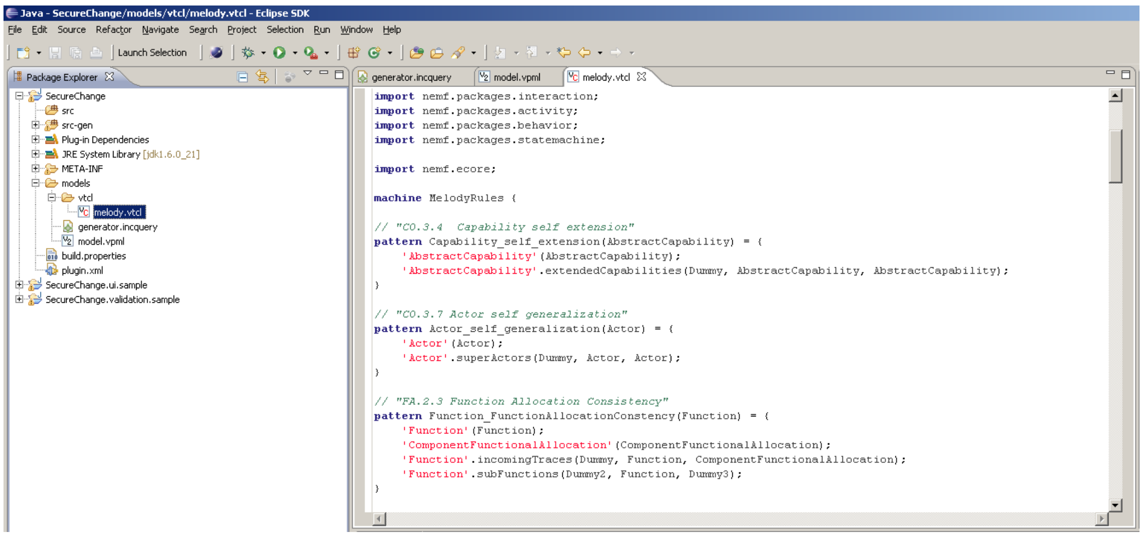
Our experiment environment consisted of the EMF-INCQUERY engine and a System engineering tool dedicated to this industrial context. This latter tool has been built on top of the Eclipse Obeo Designer tool and exposes a dedicated engineering language providing user-friendly ergonomics.

It allows engineers to define the architecture description of a software system by providing the five following views:

- The “Operational Analysis” model level, where the customer needs are defined and/or clarified in terms of tasks to accomplish by the System/Software, in its environment, for its users.
- The “System Analysis” model level, that provides a “black box” view of the System where the System limits are defined and/or clarified in terms of actors and external interfaces, the System capabilities and functional and non-functional needs and expectations; allowing to identify the more constraining/impacting requirements.
- The “Logical Architecture” model level, which provides a “white box” view of the System. It defines a technical and material independent decomposition of the System into components, and where the non-functional constraints are refined and allocated.

- The “Physical Architecture” model level, which is defined by the structuring architecture of the System. It takes into account non-functional constraints, reuses legacy assets and applies product policies.
- The “EPBS (End Product Breakdown Structure)” model level is an organizational view identifying the configuration items for development contracts and further Integration, Verification and Validation.

The EMF-INCQUERY consistency engine was integrated on top of this tool. The engine executes a set of queries representing consistency rules (specified in a VTCL file) and returns the list of detected inconsistencies to the user. Figure 17 shows a screenshot of the modelling environment and a view of the VTCL file containing the consistency rules.



```
import nemf.packages.interaction;
import nemf.packages.activity;
import nemf.packages.behavior;
import nemf.packages.statemachine;

import nemf.ecore;

machine MelodyRules {

// "CO.3.4  Capability self extension"
pattern Capability_self_extension(AbstractCapability) = {
    'AbstractCapability'(AbstractCapability);
    'AbstractCapability'.extendedCapabilities(Dummy, AbstractCapability, AbstractCapability);
}

// "CO.3.7 Actor self generalization"
pattern Actor_self_generalization(Actor) = {
    'Actor'(Actor);
    'Actor'.superActors(Dummy, Actor, Actor);
}

// "FA.2.3 Function Allocation Consistency"
pattern Function_FunctionAllocationConstency(Function) = {
    'Function'(Function);
    'ComponentFunctionalAllocation'(ComponentFunctionalAllocation);
    'Function'.incomingTraces(Dummy, Function, ComponentFunctionalAllocation);
    'Function'.subFunctions(Dummy2, Function, Dummy3);
}
```

Figure 17. EMF-INCQUERY consistency rules

C.4. Results

The lack of support for EMF derived features in EMF-INCQUERY was a major issue in our experimentation. Unfortunately our industrial meta-model uses derived features extensively. As a result, none of the consistency rules could be translated without running into one of them and, therefore, could not be run by the EMF-INCQUERY engine.

On the other hand, other merits of EMF-INCQUERY could be assessed. The declarative language, similar in expressive power and style to the Praxis Rules language used by Praxis, is well adapted to the specification of inconsistencies when compared to an imperative language like Java. In particular, such a general purpose imperative language cannot support incremental verification. In addition, Prolog-based approaches like Praxis suffer from performance limitations that EMF-INCQUERY overcomes at the cost of a higher memory usage. This should alleviate the performance issues that Thales ran into on large models with Praxis.

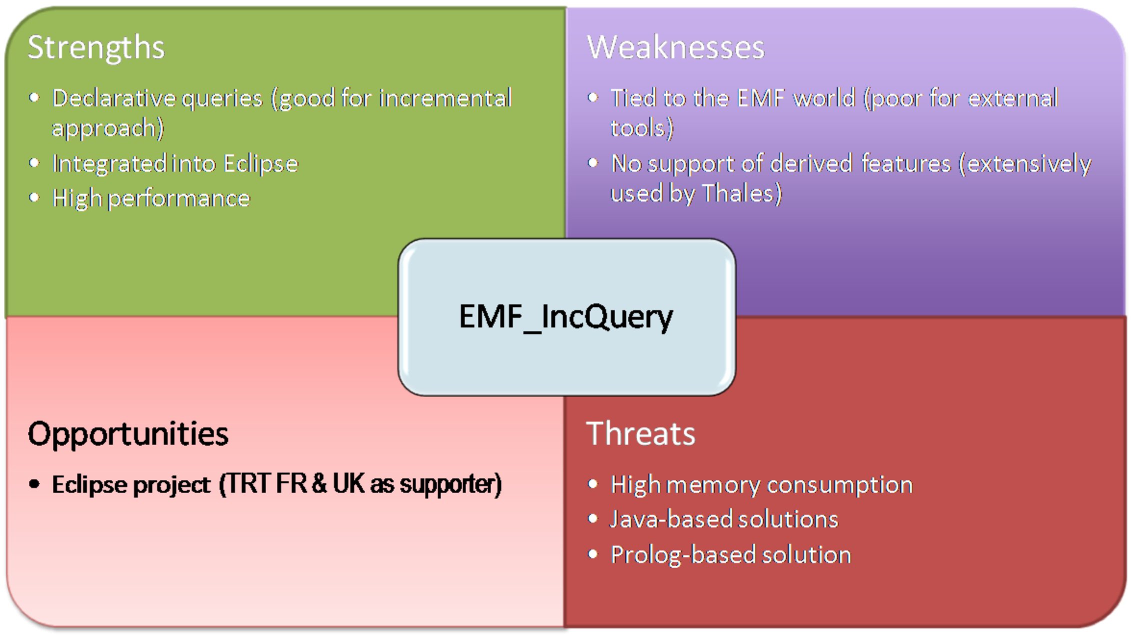

Strengths
Declarative queries (good for incremental approach)
Integrated into Eclipse
High performance
Weaknesses
Tied to the EMF world (poor for external tools)
No support of derived features (extensively used by Thales)
EMF_IncQuery
Opportunities
Eclipse project (TRT FR & UK as supporter)
Threats
High memory consumption
Java-based solutions
Prolog-based solution

Figure 18. Thales SWOT analysis of EMF-INCQUERY

Shortly after the end of Secure Change, BME has planned to release a version of EMF-INCQUERY with a limited support for derived features, which Thales will investigate.

Appendix D. Report on Second ATM Workshop

(see attachment)

Appendix E. A Tool for Managing Evolving Security Requirements

(see attachment)

Appendix F. SeCMER: A Tool to Gain Control of Security Requirements Evolution

(see attachment)

Appendix G. Orchestrating Security and System Engineering for Evolving Systems

(see attachment)