D.3.3 ALGORITHMS FOR INCREMENTAL REQUIREMENTS MODELS EVALUATION AND TRANSFORMATION

Michela Angeli (UNITN), Gábor Bergmann (BME), Fabio Massacci (UNITN),Bashar Nuseibeh (OU),Federica Paci (UNITN), Bjornar Solhaug (SINTEF), Thein Than Tun (OU), Yijun Yu (OU), Dániel Varró (BME)

Document information

Document Number	D.3.3
Document Title	Algorithms for Incremental Requirements Models Evaluation and Transformation
Version	1.19
Status	Final
Work Package	WP 3
Deliverable Type	Report
Contractual Date of Delivery	31 January 2012
Actual Date of Delivery	31 January 2012
Responsible Unit	UNITN
Contributors	OU, UNITN, BME, THA
Keyword List	
Dissemination level	PU

Document change record

Version	Date	Status	Author (Unit)	Description
1.0	30 November 2011	Draft	Federica Paci (UNITN)	Outline of the deliverable
1.1	8 December 2011	Draft	Federica Paci (UNITN)	Introduction
1.2	11 December 2011	Draft	Ábel Hegedüs (BME)	Introduction
1.3	16 December 2011	Draft	Ábel Hegedüs (BME)	Section 4
1.4	17 December 2011	Draft	Thein Tun(OU)	Section 2
1.5	19 December 2011	Draft	Federica Paci (UNITN)	Introduction, Section 3 and draft of Executive Summary
1.6	19 December 2011	Draft	Ábel Hegedüs (BME)	Introduction
1.7	23 December 2011	Draft	Thein Tun (OU)	Revised Section 2
1.8	23 December 2011	Draft	Le Ming Sang (UNITN)	Added Section 4
1.9	23 December 2011	Draft	Federica Paci (UNITN)	Final version of the deliverable
1.10	28 December 2011	Draft	Michela Angeli (UNITN)	First quality check completed-minor remarks
1.11	14 January 2012	Final	Federica Paci (UNITN)	Reviwers' Comments Addressed
1.12	15 January 2012	Final	Ábel Hegedüs (BME)	Reviewers' commentsaddressed
1.13	16 January	Final	Thein Tun	Addressed

			(OU)	reviewers' comments
1.14	19 January	Final	Federica Paci (UNITN)	Minor Changes
1.15	23 January	Final	Ábel Hegedüs (BME)	Addressed reviewers' comments
1.16	24 January	Final	Thein Tun (OU)	Addressed reviewers' comments
1.17	26 January	Final	Federica Paci (UNITN)	Addressed reviewers' comments
1.18	26 January	Final	Michela Angeli (UNITN)	Second quality check completed-minor remarks
1.19	31 January	Final	Federica Paci (UNITN)	Final version revision

Executive summary

Deliverable 3.4 presents the results of tasks *T3.2 Incremental Model Transformation* and *T3.3 Incremental V&V Change Analysis* during the third year of SecureChange project. The main focus of the activities conducted in Work Package 3 is the integrability into industry practice of the results obtained in Year 2. The main results of Year 2 are:

a) argumentation analysis to reason on the satisfaction of security requirements under evolution,

b) a quantitative reasoning which helps the designer to select a system design that is resilient to changing requirements, and

c) a semi-automatic approach to requirements change management based on incremental graph patterns and change driven transformations.

This deliverable shows how these results have been revised in order to allow the integration into existing industrial security engineering processes. The deliverable not only shows that the results of Work Package 3 can be adopted in industry but also that when they are adopted, they improve over the change management processes adopted in industry.

Index

1 Introduction

This deliverable presents the results of Work package 3 activities during the third year of SecureChange project.

According to the plan set forth in the DoW the activities of Work package 3 were focused on the integratability of the concepts and results from Y2 into the main Secure Change Process and into industry practice.

The main results of Year 2 reported in D3.2 and D3.3 were

- a qualitative and a quantitative reasoning technique for evolving requirements models, and
- a semi-automatic approach to requirements change management that is based on incremental graph patterns and change driven transformations.

The qualitative reasoning technique is based on argumentation analysis that is one of the steps of the SecMER methodology. Argumentation analysis is used to reason on security requirements satisfaction under change. Instead, the quantitative reasoning helps the designer to select a system design that is resilient to changing requirements. The reasoning is based on the modeling of the evolution as a set of rules and on the computation of two metrics called maximal belief and residual risk. Change-driven transformation based on security patterns allow to check argument validity**,** to automatically detect violations or fulfilment of security properties, and to issue alerts prompting human intervention, or potentially triggers automated reactions in certain cases.

In order to show how these results can be integrated into existing industrial security engineering processes, we have focused on the integration of the analysis of Security Requirements with the following steps of Risk Assessment and Security Testing from the perspective of existing processes.

In Section 1 we present a typical industrial security engineering process that is based on the orchestration of risk assessment and system engineering processes. In Section 2 we show that requirements evolution modelling and argumentation analysis can be orchestrated with risk assessment activities which are part of industrial security engineering processes. In Section 3 we show that requirement evolution modelling can also be orchestrated with another crucial activity in industrial security engineering processes, that is security testing.

The usefulness of both the quantitative reasoning and of change driven transformation to the change management process adopted by Air Navigation Providers has been validated during the workshops conducted with ATM experts at DeepBlue premises. The validation activity is described in appendix E of this deliverable. In Section 4 and 5 we illustrate how the quantitative reasoning technique on requirements evolution and change-driven transformation based on security patterns have been revised to address the need to have automatic decision support tools for change management in the air traffic management domain.

2 Orchestrating Security and System Engineering processes

This section presents an industrial security engineering process where the risk engineering process which has been standardized independently can be orchestrated into the overall system engineering process. The need of an orchestrated process is pointed in international standards like ISO/IEC 15288:2008 and ISO/IEC 12207:2008 [16, 17]. These standards describe the system and software life cycle of the engineering process and including clauses mentioning that non-functional properties such as security should be considered in different phases.

In security specialty engineering, risk analysis methodologies such as EBIOS, CORAS or CRAMM give the rationale for security requirements and assess the risks in an exhaustive way, as needed in domains such as administration or military systems. The risk management process does not cover the entire security engineering activities but is a key starting point to them.

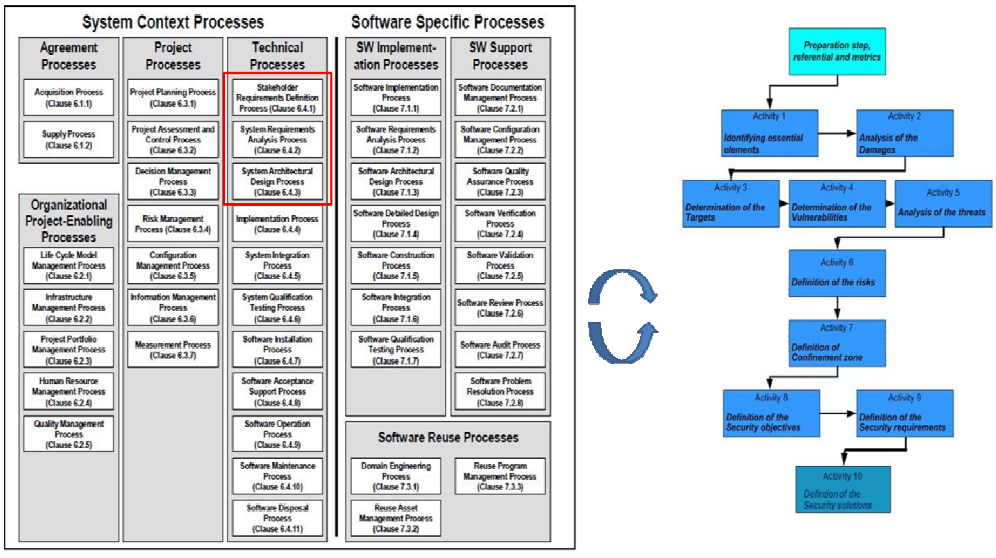


Figura 1. ISO/IEC 12207 vs EBIOS process

Thus a first issue is to show how the risk management process and security requirement analysis can collaborate with the global system engineering process described in those engineering standards. The difficulty resides in the necessary iterations needed to refine the security requirements since some vulnerabilities and risks will appear only once the system architectural design has been set up.

In particular we investigate how the processes Stakeholder Requirements Definition (Clause 6.4.1), Requirements Analysis (Clause 6.4.2), and Architectural Design

(Clause 6.4.3) processes of ISO/IEC 12207 can be orchestrated with EBIOS risk methodology activities (see Figure 1).

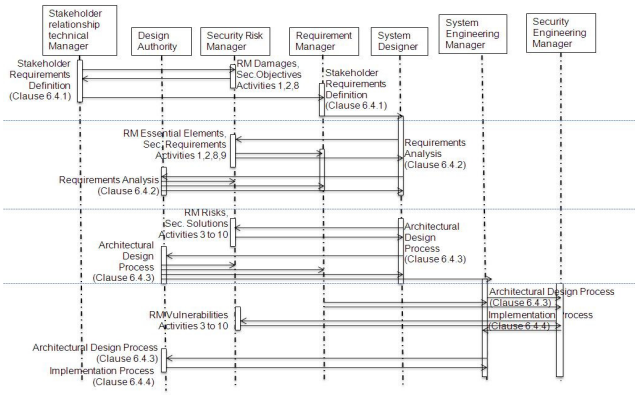


Figure 2.ISO/IEC 12207 sub-set of processes and EBIOS collaboration diagram

The resulting orchestrated process is represented in Figure 2.The stakeholder relationship1 technical manager gets the needs and requirements from the stakeholders (Clause 6.4.1). He pushes the information related to security needs to the security risk manager who expresses the unwanted damages and defines the first security objectives. The stakeholder relationship technical manager validates the security objectives with the stakeholders and consolidates them before sending them to the requirement manager. Then the requirement manager consolidates them with requirements from other stakeholders and sends all the requirements to the system designer.

The system designer analyzes the requirements (Clause 6.4.2) and defines the functions of the system. Once this is done, the security risk manager updates the essential elements (Activity 1) based on the functions of the system, updates the damages (Activity 2), adds some security objectives (Activity 8), defines first security requirements (Activity 9) and sends them to the system designer and to the requirement manager. The system designer validates the requirements analysis with the design authority who propagates it.

The system designer proceeds to architectural design of the system, allocating functions to elements of the system (Clause 6.4.3). This new organization of the model of the system is analyzed by the security risk manager who evaluates carefully the risks and defines security solutions (Activities 3 to 10), and sends the updates of the

[1] The stakeholder relationship manager in the majority of requirement engineering framework is called requirement manager

security solutions to the system designer who consolidates the system design and validates the architectural design with the design authority who propagates it. Architectural design and updated requirements are propagated to the system engineering manager and the security engineering manager for them to complete the design at physical layer and implement it (Clause 6.4.3 and Clause 6.4.4).Once he has chosen the security solutions, the security engineering manager sends the information to the security risk manager for targets and vulnerabilities determination (Activities 3 and 4) and a full update of the risk management cycle (Activities 5 to 10).

The updates are passed through the security engineering manager to the system engineering manager. The system engineering manager validates the architectural design and the existing elements of the implementation with the design authority.

3 Orchestrating Requirements and Risk Assessment processes

Changing requirements might give rise to potential security risks that in turn require some treatments to ensure and maintain an acceptable security risk level. Or treatment options that result from risk assessment may lead to new security requirements that should be included in the requirement model. Moreover, the requirement changes may involve new assets the risk level of which needs to be assessed. Thus, there is the need to trace changes to security knowledge such as assets, attacks and treatments to stakeholders' goals and security requirements and vice versa.

Current industrial software/system engineering processes are supported by artifacts (documents, models, data bases) that are disjoint and cannot be fully integrated for a variety of reasons (separate engineering domains, outsourcing, confidentiality, etc.). Thus, the collaboration between risk analyst and requirement analyst in such processes is sometimes difficult, especially when a change occurs and they have to interact to keep the risk and the requirement models mutually consistent under change.

The papers in Appendix A and B propose a possible solution to change propagation that is based on the orchestration of the requirement engineering and risk assessment processes. The orchestration relies upon mappings between key concepts of the requirement and the risk conceptual models.

In section 3.1, weinstantiate the requirement and risk conceptual models, to SI* and CORAS respectively. However, alternative approaches to requirement engineering and risk analysis with similar underlying conceptual frameworks can be orchestrated by following the same approach. Indeed, the concepts that are mapped in the former -- such as goal, resource, and task -- are in common to other goal-oriented approaches to requirement engineering. The same holds for asset and treatments that are key concepts in other risk analysis approaches. In section 3.2 we show that the same approach can be applied to SI* and Security DSML [19], which is a language and a tool developed to conduct security risk analysis in industry.

In section 3.3 we also illustrate how argumentation analysis can be orchestrated with risk assessment.

3.1 Orchestrating SI* and CORAS concepts and processes

We have mapped concepts in SI* and CORAS 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. We have identified three conceptual mappings: **ServiceToAsset**, **ServiceToTreatment** and **TreatmentToTask**.

- **ServiceToAsset**. A service that is linked to a soft goal by a “protects” relation which denotes a resource, a task or a goal that is of value for the organization and thus needs to be protected from harm. Since in CORAS, an asset is

something of value that requires protection, we map a service protected by a soft goal in SI* to an asset in CORAS.

- **ServiceToTreatment**. A service which is related to another service mapped to an asset in a CORAS model, can be mapped to a treatment if the service reduces the likelihood or the consequence of a threat scenario damaging the asset.
- **TreatmentToTask**. A treatment is a security control that should reduce the likelihood or consequence of a threat to an asset which results in a loss of confidentiality, availability, or integrity. Thus, if a treatment is implemented, the confidentiality, integrity or availability of an asset is protected. A treatment in CORAS can therefore be mapped to a task which fulfills the soft goal which specifies the security property that has to be preserved for an asset.

These are several examples of possible change propagation scenarios that are supported by these conceptual mappings. In what follows we illustrate some of these scenarios based on the introduction of a new surveillance tool, the Automatic Dependent Surveillance-Broadcasting (ADS-B) into ATM systems. ADS-B provides accurate aircraft position information by using a global navigation satellite system.

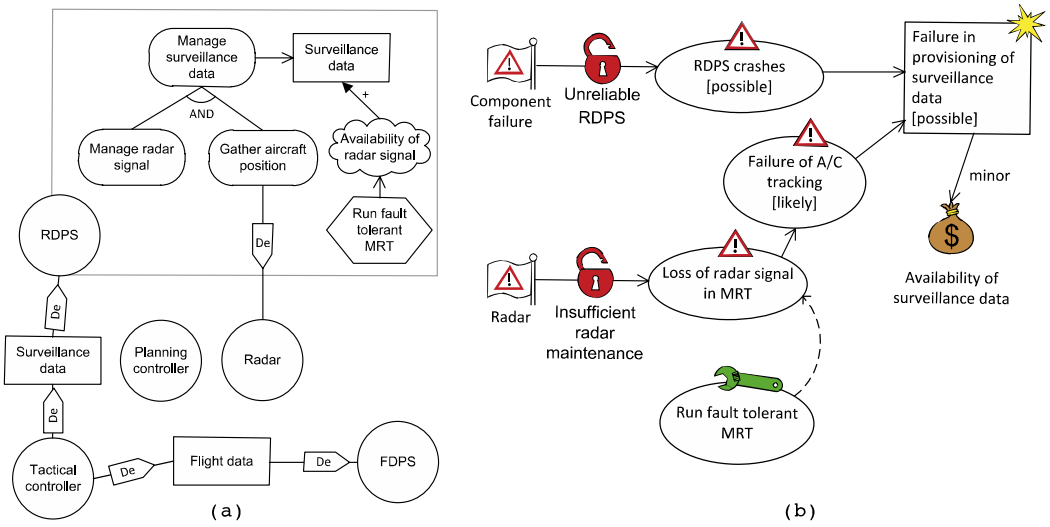


Figure 3. SI* and CORAS models before change

However, ADS-B makes ATM systems vulnerable to new threats. For example, ADS-B transmissions can be easily corrupted: a concrete example is the spoofing of the GPS satellite that provides the GPS signal to determine aircraft position.

Let's first illustrate a change propagation scenario that is based on the **ServiceToAsset** and **TreatmentToTask** conceptual mappings. The SI* and CORAS models before change are illustrated in Figure 3. The change we consider first is the introduction of a new actor, the ADS-B, which is intended to increase the accuracy and availability of the surveillance data. Figure 4 illustrates the SI* and CORAS models after the introduction of the ADS-B: the changes that are handled are highlighted in

grey. The ADS-B introduction requires the addition of the goal Manage ADS-B signal and the resource ADS-B signal. Moreover, since the integrity of the ADS-B signal is critical, introducing a new soft goal Integrity of ADS-B signal specifies this security need. Notice also that because ADS-B signal becomes part of the surveillance data, the soft goal Availability of surveillance data is decomposed into the sub goals Availability of ADS-B signal and Availability of Radar signal. The introduction of the ADS-B actor may affect the risks due to the introduction of the new soft goals. In particular, the soft goal Integrity of ADS-B signal is mapped to a corresponding asset in the CORAS model for which a risk assessment is conducted. As ADS-B is prone to data spoofing, this issue is addressed and identified as an unacceptable risk with respect to integrity. The treatment Apply MD5 checksum is considered, and leads to the addition of a new task Apply MD5 Checksum fulfilling the soft goal Integrity of ADS-B signal in the SI* model. The new soft goal Availability of ADS-B signal is also mapped to the CORAS model as a new aspect of the more general asset Availability of surveillance data. The risk analyst decides not to decompose this asset with respect to ADS-B and radar when assessing the impact of the ADS-B. A new threat scenario Loss of ADS-B signal is identified, but the overall risks with respect to availability of surveillance data do not increase; rather, the likelihood of the threat scenario Failure of A/C tracking decreases from likely to possible (see Figure 3 (b)).

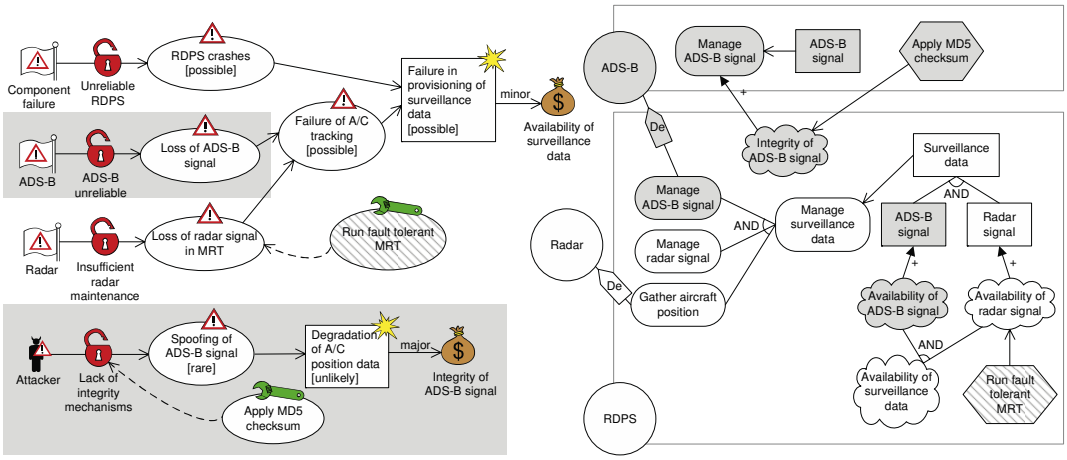


Figure 4. Post change SI* and CORAS models

Another possible scenario of change-driven interplay is based on the **ServiceToTreatment** and **TreatmentToTask** conceptual mappings. The introduction of a new actor, the ADS-B, increases the availability of surveillance data. Thus, the likelihood of the threat scenario Failure of A/C tracking in the CORAS model is reduced. Consequently, the risk analyst determines that the treatment Run fault tolerant MRT is no longer needed and therefore removes it from the CORAS model.

This treatment is mapped to the corresponding task in the requirement model since it protects the soft goal Availability of radar data, and hence the more general soft goal Availability of surveillance data. Therefore, the removal of treatment Run fault tolerant MRT in the CORAS model leads to the removal of the mapped task Run fault tolerant MRT. In Figure 4 the removal of elements is indicated by the diagonally striped elements.

The change-driven interplay scenarios are supported by the formalization of the mappings as VIATRA2 graph transformation rules. The formalization is reported in Appendix B. The execution of the rules ensures that consistency is maintained between the two models and that none of the models become obsolete with respect to the other under change.

3.2 Orchestrating SI* and Security DSML concepts and processes

Here we illustrate some of the steps of the orchestrated process based on the conceptual mappings between SI* and Security DSML shown in Table 1. The steps of the process are presented using the introduction of the AMAN as illustrative example.

	Conceptual Mapping		
Requirement	**Risk**	**Architecture**	**Type**
Business Object	Essential Element		Shared
Goal	Security Objective		Mapped
Security Goal	Security Requirement		Mapped
Process		Security Solution	Mapped

Table 1. Concepual Mapping between SI* and Security DSML

The stakeholder relationship technical manager and the security risk manager interact to identify an initial set of security objectives to be passed to the requirement manager.

The stakeholder relationship technical manager passes the requirements proposed by the stakeholders and the initial security objectives to the requirement manager. A change request is triggered for the requirement domain: the SI* model illustrated in Figure 1 is produced by the requirement manager.

The system designer analyzes the SI* model provided by the requirement manager and then passes it to the security risk manager.

The security risk manager identifies the following new security objectives:

- **O1** The system shall be computed automatically by an Arrival Manager system that covers the risk;
- **R1** Failure in the provisioning of correct or optimal arrival information due to ATCO mistakes;
- **O2** The update of the system should be handled through a dedicated role of Sequence Manager that covers the risk **R1**.

The above security objectives are refined into the following security requirements:

- **RE1** The system should integrate an AMAN (refines security objective **O1**)
- **RE2** The organization should integrate a SQM (refines security objective **O2**).

Figure 5 represents the Security DSML model updated with the new security objectives and security requirements.

The changes into the Security DSML model trigger a change request for the requirement domain. The requirement manager receives the new security objectives and requirements and updates the SI* model as shown in Figure 6: two new actors, AMAN and the SQM have been added with their goals, process and resources.

The new processes Compute Arrival Sequence provided by AMAN and Monitor and Modify provided by SQM identified by the requirement manager has to be propagated to the system designer and to the security risk manager.

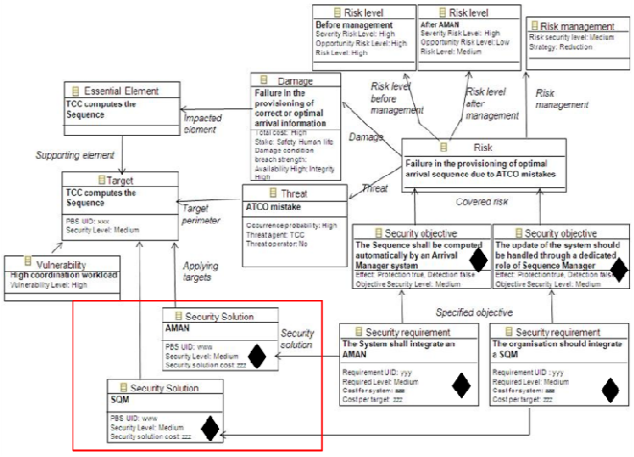


Figure 5. Security DSML Model after the introduction of AMAN

The security risk manager assesses the new processes proposed by the requirement manager and defines new security solutions to match the processes (outlined in red in Figure 5). Then, the security risk manager passes the identified security solutions to the system designer for validation.

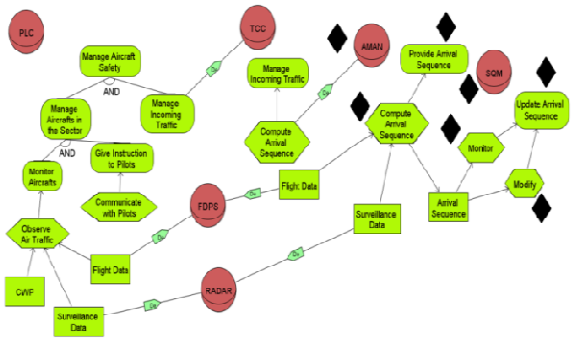


Figure 6. SI* Model after the introduction of the AMAN and SQM

3.3 Orchestrating Argumentation and Risk Assessment Processes

Argumentation analysis is one of the main steps of the SecMER methodology. Arguments provide a way to structure the system artifacts involving the concepts in the SecMER conceptual model. The requirements analysis uses the SecMER requirement model to sketch informal arguments for the security goals of the system which will be affected by the proposed change.

In this section we orchestrate argumentation with risk assessment. We introduce RISA approach to risk-based argumentation for evolving security requirements proposed in [5], and its application to the ATM case study. In the paper attached in Appendix A, RISA is applied to another case study, the PIN Entry Device (PED) system. The change scenario in the paper is the introduction of variant PED systems, and the main security properties are confidentiality and integrity of PIN.

3.3.1 Overview of the approach

Haley et al. [6] have shown that argumentation provides an appropriate framework for demonstrating that a given software system satisfies its security requirements. They separate arguments into two kinds. An outer argument is a formal proof that the software (S), within a world context (W), satisfies the security requirements (R): W, S |- R. Statements about the software and the world context are called behavioural premises, the correctness of which is critical for system security. In order to demonstrate their correctness, these statements are challenged and counter-challenged by means of inner arguments.

The diagram in Figure 7 summarises the main steps of the proposed approach, named RISA (RIsk assessment in Security Argumentation), extending the framework of Haley et al [6]. As indicated in the diagram, Steps 1 to 3 of the proposed approach are the same as the first three steps of the framework of Haley et al. Steps in shaded boxes

are supported by the argumentation tool OpenArgue [7], developed in the SecureChange project.

In Step 1 (Identify Functional Requirements), functional requirements of the system and the system context are identified. These requirements may be derived from the higher-level goals of the system. In Step 2 (Identify Security Goals), assets that need to be protected, and security goals are identified. In Step 3 (Identify Security Requirements), security requirements are derived from security goals, and are expressed as constraints on the functional requirements identified in Step 1.

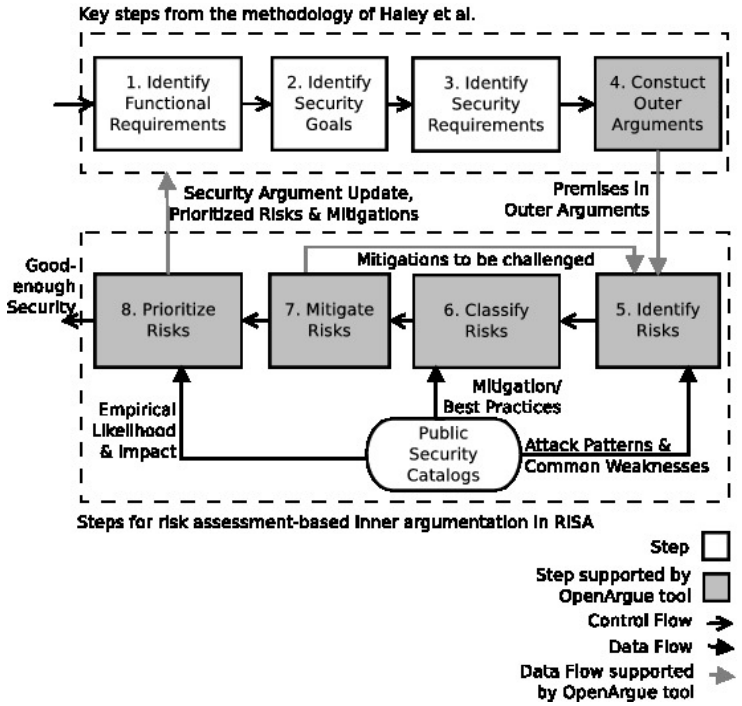


Figure 7. Overview of the RISA approach

Unlike the fourth step of the Haley et al. framework, only the outer arguments for security requirements, excluding the inner arguments, are constructed in Step 4 (Construct Outer Arguments) of RISA. These outer arguments are formal, and they make use of domain properties, correctness of which is examined by inner arguments. Behavioral premises used in the outer arguments may contain risks, which are identified as part of risk assessment in RISA.

In Step 5 (Identify Risks), behavioral premises in outer arguments are analyzed in terms of potential risks that rebut the premises. For instance, in the PED example, there could be a behavioral premise about the confidentiality of the PIN entered using the PED keypad. Public security catalogues are then searched to find known security weaknesses and attack patterns regarding the confidentiality of PIN entered by consumers using a keypad.

In Step 6 (Classify Risks) the catalogue entries related to risks identified in the previous step to (i) find appropriate security mechanisms for mitigating them and (ii) classify

these mechanisms, and indirectly the risks associated with them, according to whether the system or the context should mitigate the risks.

In Step 7 (Mitigate Risks), only the risks with mitigations assigned to the system are considered. It involves the consolidation of mitigations reoccurring in several risks. It consists of (1) numbering mitigations, (2) assigning to each of them a list of risks it rebuts, and (3) updating their description to comply with all these risks, if applicable. Some of these mitigations, themselves, could introduce new risks and therefore should be assessed in a new round of inner argumentation. In the last step (Prioritize Risks), risks are prioritized on the basis of their severity as indicated by the public security catalogs (arrow from catalogs to Step 8). These risks affect the priority of requirements to be satisfied (arrow from Step 8 to Steps 1–4). When the residual risks are deemed to be acceptable given the limitation of development resources, the system has reached the level of good-enough security.

3.3.2 Application to the ATM case study

We now briefly illustrate the application of the RISA approach to the (Automatic Dependent Surveillance-Broadcasting) ADS-B introduction scenario of the ATM case study. The purpose of the discussion is to briefly highlight how the approach will be applied to the evolutionary ATM scenario used in the report by presenting the main input and output from the steps.

***Step 1 (Identify Functional Requirements)*:** We begin by developing the after change requirement model. In the case of ADS-B introduction, the functional requirement (or the goal) is to manage surveillance data. As shown in the diagram in Figure 8, the "Manage Surveillance data" goal can be decomposed further into two subgoals in the after model.

***Step 2 (Identify Security Goals)*:** One overall security/soft goal is to ensure the availability of surveillance data. This is a system level goal that has to be decomposed into a number of requirements for various parts of the system. The main assets after the change are Surveillance data, ADS-B signal and Radar signal, as shown in Figure 8. Another security goal is the integrity of ADS-B signal, which is not discussed further here.

Step 3 (Identify Security Requirements): Applying the security goal of "Availability to surveillance data" to the functional requirements gives a new security requirement, that is ensuring the "Availability of ADS-B surveillance data".

Step 4 (Construct Outer Arguments**):** The main argument for the availability of ADS-B surveillance data in the after change scenario will take of the following form:

$$P1, P2, P3 \rightarrow P4$$

where P1 is "Planes are fitted with ADS-B devices", P2 is "GPS signals are available", P3 is "Ground surveillance system is fitted with ADS-B receivers" and P4 is "ADS-B surveillance data is available". (See Appendix C for formalisation of arguments.)

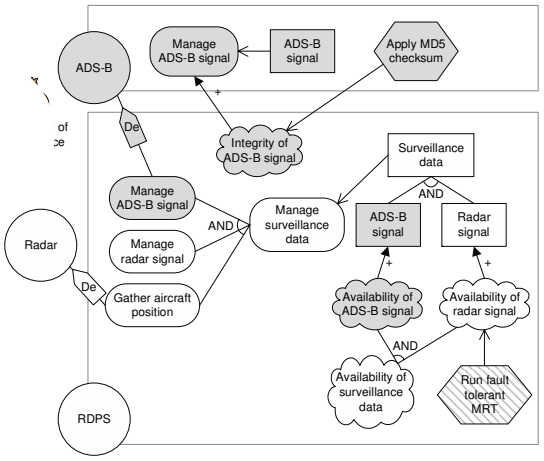


Figure 8. Requirement model for ADS-B Introduction

***Step 5 (Identify Risks)*:** There are several risks associated with each of the behavioural premises (P1, P2 and P3). For instance, GPS signal availability may be at risk due to several forms of attack, such as ADS-B signal jamming and the ADS-B system turn off. Here the method user can adopt an appropriate risk identification technique such as the one proposed by CORAS.

***Step 6 (Classify Risks)*:** ADS-B signal jamming is a risk that may be transferred to the context, whilst the ADS-B system turn off is a risk that may be transferred to the system.

***Step 7 (Mitigate Risks)*:** The risk of ADS-B signal jamming may be mitigated by ensuring that there is no 1090 MHz jamming near ground stations, for instance. The risk of ADS-B system turn off may be mitigated by providing no in-flight turn off functionality.

Step 8 (Prioritize Risks): Domain experts help prioritizing the risks: in this case, they are both identified as critical risks. Again, CORAS or another appropriate risk prioritization technique can be used to prioritize these risks.

4 Orchestrating Requirements and Testing Processes

In this section we give a quick overview of the orchestration of requirements engineering process and testing process. The process can be understood as an instantiation of the more general and project wide integration presented in deliverable D2.2. The process illustrated in Figure 9 is exemplified in Appendix D based on the change requirement "Specification Evolution" of the POPS case study.

The orchestration of the requirements engineering process and the test generation process is based on the identification of a set of concepts that are shared or mappable in the two domains: a *shared concept* is a concept that has the same semantics in both domains while a *mappable concept* is a concept that is related to one in the other domain. Table 2 illustrates the conceptual interface. When a concept is changed in a model then a corresponding change request is issued to the other model.

Requirement concept	**Testing concept**	**Kind of integration**
Goal	Test Model (State Machine, OCL code)	Mapped concept
Action	Test Model (State Machine, OCL code)	Mapped concept
Achievement level	Test result	Mapped concept
Actor	SUT	Mapped concept
Requirement	Requirement	Shared concept

Table 2. Conceptual mapping

We identify one shared concepts that is *Requirement*. A Requirement in both domains represents a statement by a stakeholder about what the system should do.

The concepts of *Actor, Goal* and *Process,* are mapped on the Test Model. In particular, the concept of Actor is used to identify the system under test (SUT). The concepts of Goal and Process are used by the testing engineer to build the different types of Test Models. The goals and processes in the Requirement Model are identified by a unique name that is used to annotate the State Machine of the Test Model and the OCL code in order to achieve traceability between the Requirement Model and the Test Model.

Mapping of a test case's result and status to a requirement achievement level allows the requirement engineer to quantify the requirement *coverage* after evolution. This correspondence is reported in Table 3: if the status of a test case after evolution is *new*, *adapted* or *updated*, and the test result is *pass* the requirement covered by the test case is fulfilled while it is denied (i.e. we have evidence that has not been achieved) if the test result is fail. A subtle case is present when a test case is part of

the stagnation suite (i.e. *obsolete*) and the test result is *fail*. Here the test covers requirements that have been deleted from the model and thus the corresponding behavior should no longer be present (for example a withdrawn authorization) so failing the test shows that the unwanted behavior is no longer present.

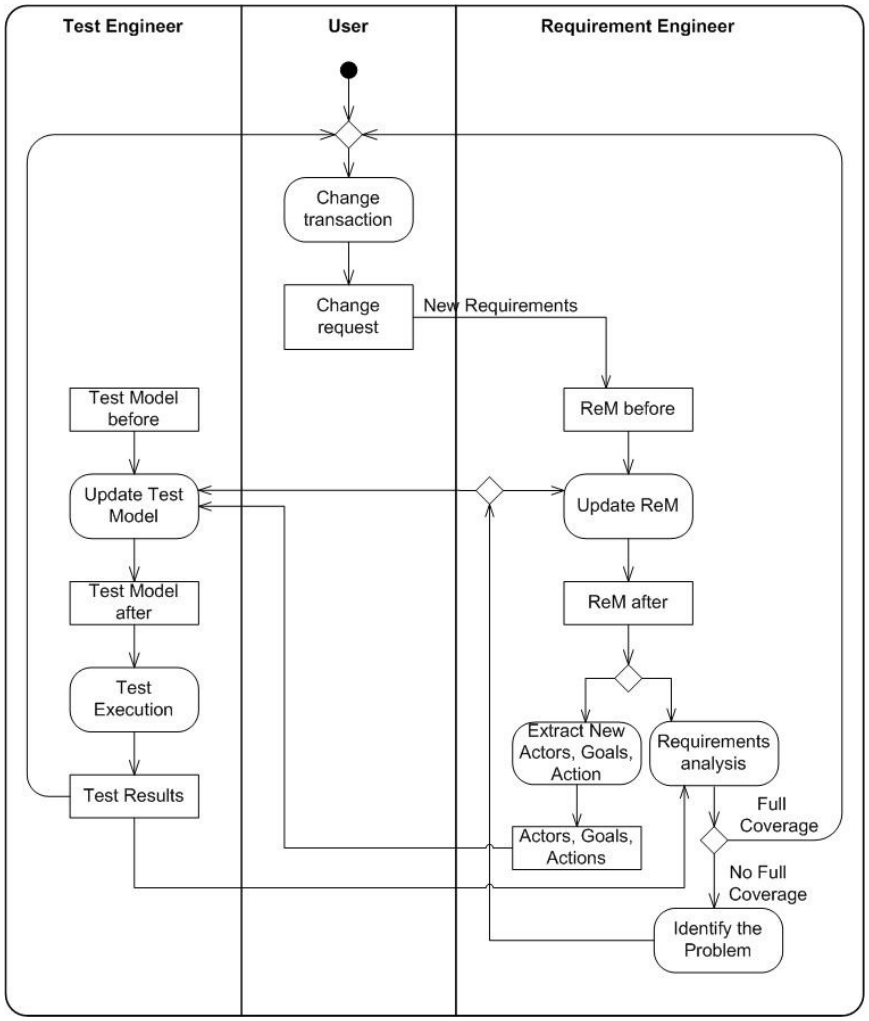


Figure 9. Orchestrated process

Test Classification	Test Status	Test Result	Achievement Level
Evolution	New, Adapted,Updated	Pass	Fulfill
Regression	Unimpacted, Re-Executable	Pass	Fulfill
Evolution	New, Adapted,Updated	Fail	Deny
Regression	Unimpacted, Re-Executable	Fail	Deny
Stagnation	Outdated, Failed	Pass	Deny
Stagnation	Outdated, Failed	Fail	Deny

Table **3. Requirements coverage**

We also consider *completion* indicators for the change propagation process which indicates whether all changes in the requirement model have been propagated to the test model. Table 4 summarizes the mapping between Goal and Process in the requirement model and the Test Model element. In a nutshell we say that the change propagation process has been completed if:

- for each new or modified model element in the ReM model a new test case and an adapted are added to the evolution test suite,
- for each model element not impacted by evolution there is a re-executable test case in the regression test suite,
- for each model element deleted from the model there is an obsolete test case in the stagnation test suite.

Change in ReM Model	Test Status	Test Suite
New Goal or Process	New	Evolution
Modified Goal or Process	Adapted	Adapted
Model Element Not Impacted	Re-Executed	Re-Executable
Deleted Element	Obsolete	Obsolete

Table **4. Completion of change propagation**

5 A framework for modeling and reasoning on goal models evolution

In this section we present an enhanced version of the framework for modelling and reasoning on evolution of requirements models which was proposed in the previous version of this deliverable submitted at M24. The framework is based on the idea of modelling evolution of requirement models in terms of two kinds of evolution rules: *controllable* and *observable rules*. The reasoning is based on the computation of two quantitative metrics called *maximal belief* and *residual risk* that intuitively measure the usefulness of a design alternative (or a set of elements) after evolution. In fact, the maximal belief tells whether a design alternative is useful after evolution, while residual risk quantifies if a design alternative is no longer useful. The framework during the third year of the project has been applied to goal models. In this section we will present an optimal algorithm to compute maximal belief and residual risk metrics for goal models.

5.1 Reasoning on goal models evolution

To illustrate the calculation of Max Belief and Residual Risk, let's consider the introduction of the AMAN as example. The critical mission of AMAN is to support ATCOs to manage the arrival traffic safely and efficiently, e.g. by maintaining an appropriate separation between aircrafts. The AMAN calculates an optimal arrival sequence considering many constraints such as flight profiles, aircraft types, airspace and runway condition, inbound flow rate, as well as meteorological conditions. The final sequence is approved by ATCOs. Then, the AMAN generates various kinds of advisories for ATCO to send to pilots e.g. time to lose or gain, top-of-descent, track extension, while their execution is continuously monitored to react to possible violations. The sequence and other relevant data are exchanged with adjacent sectors to improve collaboration and reduce conflicts.

This high level goal of AMAN is refined to many other subgoals, as illustrated by the hypergraph in Figure 10. The hypergraph consists of nodes (rounded rectangles) that represent goals to be achieved and circles that represent the AND decomposition of goals. Here the example only focuses on the sub goal '*G2- optimal arrival sequence applied*' and '*G5- Data exchanged*'. The former goal concerns the generation and application of an optimal arrival sequence with respect to a given situation. The latter goal is to exchange data for collaborating with other units.

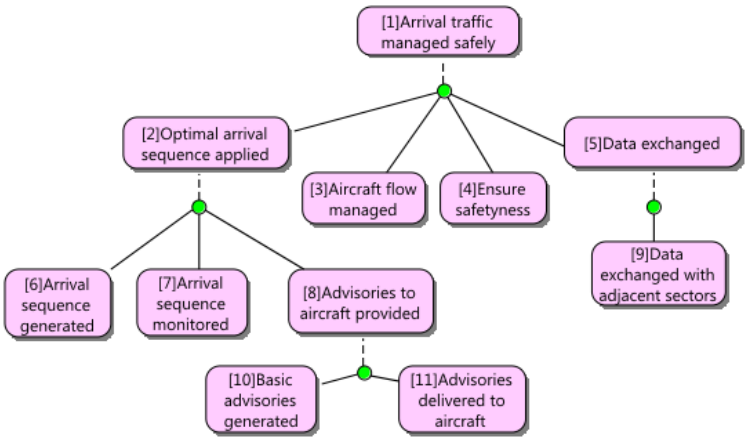


Figure 10. Goal model for AMAN's top goal 'Arrival traffic managed safely'.

The ATM working environment is continuously evolving due to numerous causes such as the increase of traffic load, the new performance and safety standards, or the need of tighter collaboration between ATM systems. These organizational level changes introduce many potential evolutions on AMAN such as:

- AMAN should support what-if-probing and online simulation.
- AMAN should support trajectory prediction, i.e. Expected Time of Arrival
- AMAN should support advanced advisories generation: heading, speed instructions.
- AMAN should be interoperable and optimized at European level, by supporting SESAR 4D operations such as Controlled Time of Arrival (CTA) and Required Time of Arrival (RTA) negotiation.
- AMAN should be able to exchange data with other queue management tool such as the Departure Manager (DMAN), or other AMANs in same or other airports.
- AMAN should integrate with other monitoring and conflict detection tools such as Monitoring Aids (MONA), Medium Term Conflict Detection (MTCD).

These organizational level changes can be represented as a set of observational evolution rules. The goal G_5 *–Data Exchanged* in Figure 5 is refined into one subgoal G_9 *–Data exchanged with adjacent sectors*. One potential evolution for is that a regulatory body requires that the organization should also achieve both goal G_{12} *- Basic data exchanged with DMAN* and goal G_{13} *-Advanced data exchanged with DMAN* albeit for different functionalities. If this possibility actually happens the organization should meet {G_9, G_{12}, G_{13}} in order to meet G_5. However, the discussion in the regulatory body is still quite open and another option might be to actually leave partners to chose whether to impose G_9 *–Data exchanged with adjacent sectors* and G_{12} *- Basic data exchanged with DMAN* and leave operators the choice of implementing G_9 and G_{13} *-Advanced data exchanged with DMAN*. These possibilities exclusively happen with some uncertainties. For example, there is 12% of probability that goal G_5*- Data exchanged* is refined to goal G_9*- Data exchanged with adjacent*

sectors. And there is 42% of probability that goal G_5 is refined to either goals {G_9, G_{12}}, or goals {G_9, G_{13}}. Finally, there is 46% of probability that goal G_5 is refined to goals {G_9, G_{12}, G_{13}}.

{G_9, G_{12}}, {G_9, G_{13}} and {G_9, G_{12}, G_{13}} are called *design alternatives* for goal G_5.

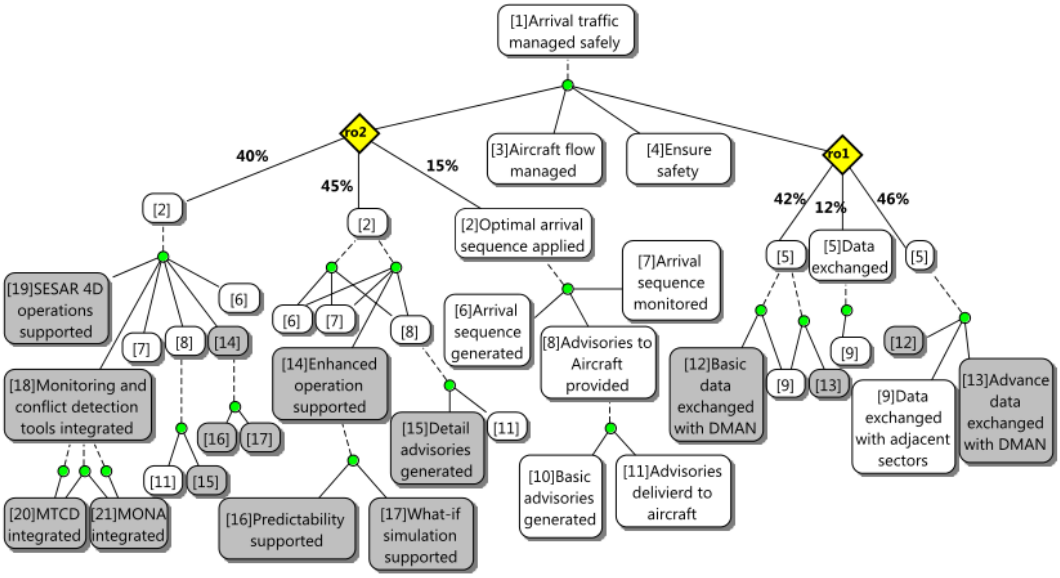


Figure 11. The goal model of Figure 5 with potential evolutions. Shaded goals denote potential changes in future, and the diamond nodes denote different possibilities that a goal may change.

In order to compute the maximal belief and the residual risk, we transform the hypergraph in Figure 10 into the hypegraph that includes goals and the possible evolution rules illustrated in Figure 11. The hypergraph includes two evolution rules for goal G_2 - *Optimal arrival sequence applied* and goal G_5 *–Data Exchanged*. Notice that, goals with a same number refer to the same objective, and only one of them is fully labeled to save the space. White goals indicate goals existing before evolution, while gray goals denote goals introduced when evolution happens. As already described G_5 might evolve to either {G_9, G_{12}, G_{13}} and {G_9, G_{12}} or {G_9, G_{13}} with a probability of 46% and 42%, respectively. The original part might stay unchanged with a probability of 12%. Similarly, goal G_2 also evolves to two other possibilities with probabilities of 45% and 40%. It stays the same with a probability of the 15%.

Each node in the hypergraph is associated with a data structure called *design alternative table* (DAT). The DAT is a set of tuples $\langle S, mb, rr, T_i \rangle$ where S is a set of leaf goals necessary to fulfill this node; *mb, rr* are the maximum belief and residual risk of this node, respectively; T_i is a possible design alternative in the observable evolution rule associated with the node. Obviously, two tuples in a DAT which have a same T_i are two design alternatives of an observable evolution possibility. For a leaf node L, the DAT of L has only one row which is $\langle \{L\}, 1, 0, \emptyset \rangle$. The DATs of leaf nodes are then propagated upward to predecessor nodes (ancestors). This propagation is done by two operators *join* ($\otimes$) and *concat* ($\oplus$). Also via these operators, DAT of a predecessor node is generated using their successor's DATs. *join* ($\otimes$) and *concat* ($\oplus$) are defined as follows.

$$DAT(x) \oplus DAT(y) = DAT(x) \cup DAT(y)$$

$$DAT(x) \otimes DAT(y) = \{\langle S_x \cup S_y, mb_x \times mb_y, 1 \quad (1 \quad rr_x)(1 \quad rr_y), T_x \cup T_y \rangle\}$$

When propagated upward, depend on the kinds of predecessor nodes and the kinds of connections among nodes, suitable operation (*join* or *concat*) will be applied. *join* is used to generate the DAT of a compound node where the semantic is that all child node are chosen. Meanwhile, *concat* is used to generate the DAT of a goal node or observable node where the semantic is the selection of one among its successor.

$$DAT(x) \leftarrow \begin{cases} \oplus_{\forall \langle g_i, x, p_i \rangle} \left(map_{\langle x, i, p_i \rangle} DAT(g_i) \right), x \text{ is an observable node} \\ \oplus_{\forall \langle c_i, x, 1 \rangle} DAT(c_i), \quad x \text{ is an goal node} \\ \otimes_{\forall \langle y_i, x, p_i \rangle} DAT(y_i, x, p_i), x \text{ is a compound node} \end{cases} \quad (1)$$

Where $map_{\langle x,i,p_i \rangle} DAT(g) = \cup_j \{\langle S_j, p_i \times mb_j, 1 - p_i(1 - rr_j), \langle x, i \rangle | T_j \rangle\}$. The operator '|' denotes the string concatenation operator e.g. a | {b, c} = {ab, ac}. The Max Belief and Residual Risk of a design alternative are calculated by following equation.

$$\begin{aligned} MaxBelief(C) &= \max_{\forall \langle S_i, mb_i, rr_i, T_i \rangle \in SDA(C)} mb_i \\ ResidualRisk(C) &= 1 - \sum_{\forall \langle S_i, mb_i, rr_i, T_i \rangle \in SDA(C)} (1 - rr_i) \end{aligned} \quad (2)$$

Where *SDA(C)* is a subset of the DAT of the root node such that *C* is able to support. Notice that two or more tuples in a DAT which have a same *T_i* determine that they are design alternatives fulfilling a same observable evolution possibility. Thus, when calculating residual risk, only one of them is taken into account.

The comparison criterion between two design alternatives based on Max Belief and Residual Risk is that *the higher Max Belief and lower Residual Risk design alternative is the better.* If two design alternatives DA_1 and DA_2, which Max Belief of DA_1 is greater than DA_2, but the Residual Risk of DA_2, are less than that of DA_1, then we can compare DA_1 and DA_2 based on the combination of Max Belief and Residual Risk. A simple combination of these values is the harmonic mean. Hence, if the harmonic mean of Max Belief and Residual Risk of DA_1 is greater than DA_2, then DA_1 is superior in terms of Max Belief and Residual Risk.

Table 5 reports the DAT of the root node, the goal *G_1 – Arrival Traffic Managed Safely* in the hypergraph in Figure 11. Based on this table, we can compute the Max Belief and Residual Risk of any design alternative as denoted in Equation 2.

For example, given a design alternative C = {G9,G12,G6,G7,G11,G15,G16,G17,G19,G20}, the Max Belief and Residual Risk of C are computed as follows.

- **Identify design alternatives supported by C**. The design alternatives outlined in bold in Table 1 DA_2, DA_3, DA_4, DA_8, DA_9, DA_{10} are the design alternatives supported by design alternative *C*.

  $$SDA(C) = \{DA_2, DA_3, DA_4, DA_8, DA_9, DA_{10}\}$$

- **Remove alternatives that have duplicated trail (T).** Among selected alternatives, DA_2, DA_3 have a same trail, and DA_8, DA_9 also have the same trail. Thus, we remove DA_3, DA_9 from SDA(C)

$$SDA(C) = \{DA_2, DA_4, DA_8, DA_{10}\}$$

- **Calculate the Max Belief and Residual Risk.**

  Max Belief(C) = max{0.054, 0.048, 0.189, 0.168} = 0.189

  Residual Risk(C) = 1 – (1 – 0.946) + (1 – 0.952) + (1 – 0.811) + (1 – 0.832) = 0.541

Similarly, we can compute the Max Belief and Residual Risk of all design alternatives as follows.

	Design Alternative (DA)	MB	RR	Trail (T)
DA_1	$G_9,G_6,G_7,G_{10},G_{11}$	0.018	0.982	$\{\langle ro1,0\},\langle ro2,0\rangle\}$
DA_2	**$G_9,G_6,G_7,G_{11},G_{15}$**	**0.054**	**0.946**	**$\{\langle ro1,0\rangle,\langle ro2,1\rangle\}$**
DA_3	**$G_9,G_6,G_7,G_{10},G_{11},G_{15},G_{16},G_{17}$**	**0.054**	**0.946**	**$\{\langle ro1,0\rangle,\langle ro2,1\rangle\}$**
DA_4	**$G_9,G_6,G_7,G_{10},G_{11}, G_{15},G_{16},G_{17},G_{19},G_{20}$**	**0.048**	**0.952**	**$\{\langle ro1,0\rangle,\langle ro2,2\rangle\}$**
DA_5	$G_9,G_6,G_7,G_{10},G_{11}, G_{15},G_{16},G_{17},G_{19},G_{21}$	0.048	0.952	$\{\langle ro1,0\rangle,\langle ro2,2\rangle\}$
DA_6	$G_9,G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20},G_{21}$	0.048	0.952	$\{\langle ro1,0\rangle,\langle ro2,2\rangle\}$
DA_7	$G_9,G_{12},G_6,G_7,G_{10},G_{11}$	0.063	0.937	$\{\langle ro1,1\rangle,\langle ro2,0\rangle\}$
DA_8	**$G_9,G_{12},G_6,G_7,G_{11},G_{15}$**	**0.189**	**0.811**	**$\{\langle ro1,1\rangle,\langle ro2,1\rangle\}$**
DA_9	**$G_9,G_{12},G_6,G_7,G_{11},G_{15},G_{16},G_{17}$**	**0.189**	**0.811**	**$\{\langle ro1,1\rangle,\langle ro2,1\rangle\}$**
DA_{10}	**$G_9,G_{12},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20}$**	**0.168**	**0.832**	**$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$**
DA_{11}	$G_9,G_{12},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{21}$	0.168	0.832	$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$
DA_{12}	$G_9,G_{12},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20},G_{21}$	0.168	0.832	$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$
DA_{13}	$G_9,G_{13},G_6,G_7,G_{10},G_{11}$	0.063	0.937	$\{\langle ro1,1\rangle,\langle ro2,0\rangle\}$
DA_{14}	$G_9,G_{13},G_6,G_7,G_{11},G_{15}$	0.189	0.811	$\{\langle ro1,1\rangle,\langle ro2,1\rangle\}$
DA_{15}	$G_9,G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17}$	0.189	0.811	$\{\langle ro1,1\rangle,\langle ro2,1\rangle\}$
DA_{16}	$G_9,G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20}$	0.168	0.832	$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$
DA_{17}	$G_9,G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{21}$	0.168	0.832	$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$
DA_{18}	$G_9,G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20},G_{21}$	0.168	0.832	$\{\langle ro1,1\rangle,\langle ro2,2\rangle\}$
DA_{19}	$G_9,G_{12},G_{13},G_6,G_7,G_{10},G_{11}$	0.069	0.931	$\{\langle ro1,2\rangle,\langle ro2,0\rangle\}$
DA_{20}	$G_9,G_{12},G_{13},G_6,G_7,G_{11},G_{15}$	0.207	0.793	$\{\langle ro1,2\rangle,\langle ro2,1\rangle\}$
DA_{21}	$G_9,G_{12},G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17}$	0.207	0.793	$\{\langle ro1,2\rangle,\langle ro2,1\rangle\}$
DA_{22}	$G_9,G_{12},G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20}$	0.184	0.816	$\{\langle ro1,2\rangle,\langle ro2,2\rangle\}$
DA_{23}	$G_9,G_{12},G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{21}$	0.184	0.816	$\{\langle ro1,2\rangle,\langle ro2,2\rangle\}$
DA_{24}	$G_9,G_{12},G_{13},G_6,G_7,G_{11},G_{15},G_{16},G_{17},G_{19},G_{20},G_{21}$	0.184	0.816	$\{\langle ro1,2\rangle,\langle ro2,2\rangle\}$

Table 5. The DAT of the root node of the hypergraph in Figure 11.

6 Automatic Generation of Change Reactions

The SeCMER methodology includes a lightweight automated analysis step that evaluates requirements-level compliance with security principles. These security principles are declaratively specified by an extensible set of *security patterns*. A security pattern expresses a situation (a graph-like configuration of model elements) that leads to the violation of a security property. Whenever a new match of the security pattern (i.e. a new violation of the security property) emerges in the model, it can be automatically detected and reported. The specification of security patterns may also be augmented by automatic reactions (i.e. templates of corrective actions) that can be applied in case of a violation to fix the model and satisfy the security property once again.

This section presents an inductive mechanism for automated change reactions generation. The generated reactions to changes can be recorded to be used later when similar security violations are detected.

6.1 Inductive mechanisms to automatic change reactions generation

Design space exploration is the evaluation of design alternatives based on some constraints and parameters to find optimal solutions. Model transformation and incremental pattern matching can be combined with design space exploration to generate model manipulation sequences that lead to a preferred model state. In the case of requirements engineering with the SecMER tool, it can calculate different change reactions consisting of multiple manipulation steps in order to remove a violation and reach the fulfilment of security goals (see deliverable D3.3 delivered at M24). Both the desired and undesired situations related to goals and violations, respectively, can be described as graph patterns. Furthermore, the incremental evaluation and automatic detection of violations and fulfilment is also possible.

In some cases a violation introduced by a manual change cannot be trivially corrected, therefore it is possible to support the user by computing violation corrections generated by an automated process. These corrections possibilities are then evaluated by the user [4].

Change reaction can be computed by systematically exploring the design space of the model using a model-driven framework (see Figure 12). The inputs of the exploration are the instance model (and a selected model element in the violation), the goals that can be fulfilled, and the set of change operations.

The design space exploration framework first queries the goals over the instance models to find violations, then attempts to execute change operations in the context of the violations. O_p represents the list of applicable operations, while V_e is the context of the violations. At each explored model state, the goals are evaluated again and the sequence of change operations (leading to the given state) together with the

application contexts is saved if the goals are fulfilled. These sequences are complex change reactions, which are evaluated by the user and applied on the model if selected.

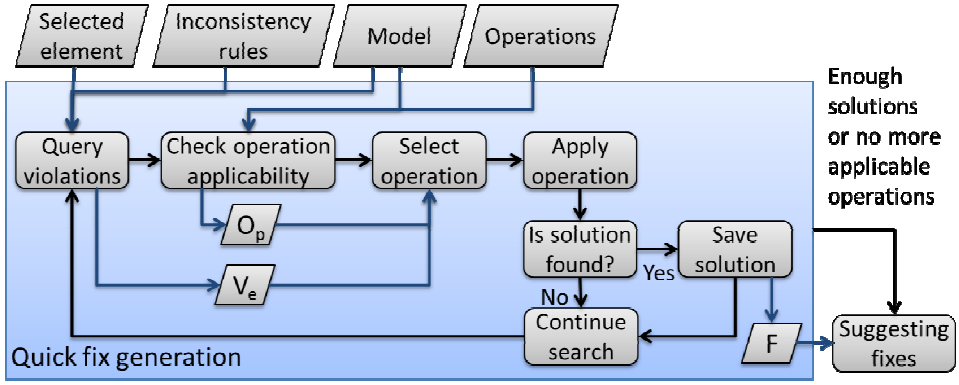


Figure 12. Quick fix generation algorithm

The set of change operations in requirement engineering includes adding new actors, creating or removing trust relations etc. These operations can be defined using model transformation rules, each including a precondition and an action part. The precondition restricts the applicability of the operation (such as a trust relation can be created only between existing elements with the appropriate type).

Once the user selects one of the generated quick fixes to be applied on the model, there is also a possibility of storing the quick fix as a complex change reaction. In order to create a reusable change reaction, its signature must be defined. The signature contains the list of elements that can be used as an application context for the change reaction. When applying the recorded change reaction using a given signature, undefined elements and variables not present in the signature but used in the sequence of operations are selected using pattern matching.

Apart from the design space exploration approach, it is also possible to record change reactions for future use by recording manual changes performed by the user on in the editor. In this case, the framework could use inductive reasoning to determine the signature for the recorded change reaction.

Further enhancement to the approach could include an automated initialization of capturing when a given event occurs (such as the appearance of a violation), then each performed action would be recorded until the violation is corrected. Thus the recording can be controlled using evolution triggers.

This way of learning complex change reactions that can be used to respond to violations and evolution triggers leads to an inductive framework, where each change reaction further enhances the response capabilities of the quick fix generation tool, and increases the usefulness of this feature.

6.2 Application to the ATM case study

Since [4] used a different case study, we now illustrate briefly how the inductive approach can be used in the ATM case study, in particular the introduction of the AMAN. We consider the starting situation where System Engineer has delegated permission on the CWP including Flight Data and State Flight Info, but its tasks or goals use only the CWP software part (see the left side of the figure below). In order to fulfil the least privilege principle, a change reaction might perform a modification of permissions to delegate only the required parts of CWP.

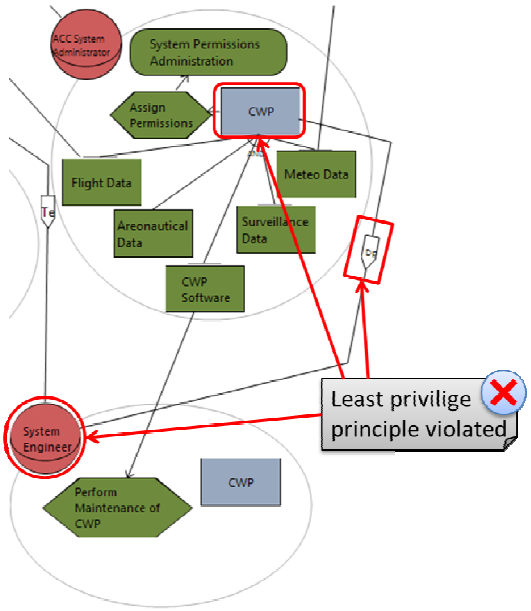


Figure 13. ATM model state before quick fix application

The goal of the quick fix generation in this case is to check that an actor has been delegated the permission to data only if it is used in at least one of its tasks or goals. The exploration would apply change operations for removing and creating delegation relations until there is no violating delegation. An example generated inductive rule is the following:

```
quick fix rule correctLeastPriviligeViolationOnData(D,RDd,A) {
        precondition leastPriviligeViolationOnData(D,RDd,A);
        action {
                call substituteDataPriviligeToParts(D,RDd,A);
                call removeUnusedDataPriviligeOnActor(A);}
```

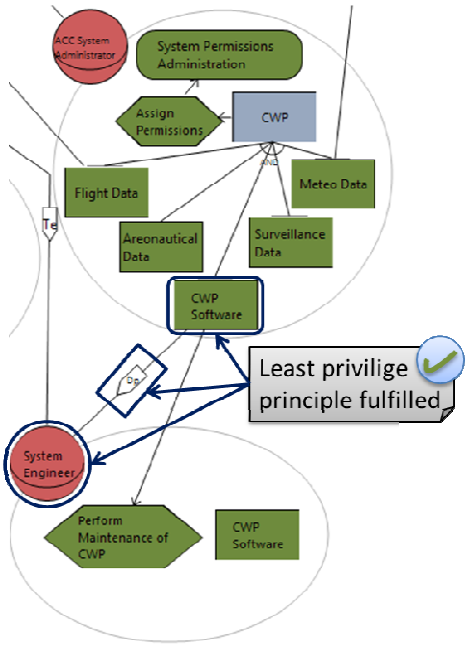


Figure 14. ATM model state after quick fix application

A possible solution model is demonstrated on the **Figure 14**. Once either the engineer or the exploration found the complex change reaction that corrects the least privilege violation, the framework can store it for future use. In this case, the precondition of the reaction is the goal, and the signature is the violation. The included simple operations replace the single delegation of permission with similar relations on the subparts, and finally remove the relations that are not needed since the data is not used.

7 CONCLUSIONS

The objective of Work Package 3 is to develop the concepts and basic building blocks for the management of evolving requirements. During the first and second year of the project, the activity of Work Package 3 has produced two main artefacts: the SeCMER conceptual model for evolving requirements, the SeCMER methodology for evolving requirements which have been presented in D3.2 and a first version of the SeCMER tool which supports the methodology steps presented in D3.4 delivered at M24. During the third year of the project, Work Package 3 activities have focused on the integrability of the results of Year 2 into industry practice. The results of the activity for integrability of Work Package 3 results into industrial processes have been documented in this deliverable. The results can be applied into industrial processes and can facilitate/improve the change management process.
In addition to that, the SeCMER tool has been improved based on the feedbacks collected during the validation activities with ATM experts. The tool interface has been made user-friendly and the ability to detect new types of security violations such as need to know principle violation has been implemented. The enhanced version of the tool is presented in deliverable D3.4 delivered at M36.

The results of Work Package 3 have contributed to advance the state-of-the art on requirements evolution management. In particular, the SeCMER methodology helps the requirement analysts in managing changes in requirements model by means of decision support artefacts and tools such as change driven transformation and argumentation analysis. Change driven transformations provide automatic detection of requirement changes and violation of security properties, while argumentation analysis helps to check whether security properties are preserved by evolution and to identify new security properties that should be taken into account. Compared with other academic and industrial tools for requirements management, the SeCMER tool provides decision support to the requirement analyst 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.

References

[1] Yudis Asnar, Fabio Massaci, Federica Paci, Bjornar Solhaugh. “A Change Driven Interplay between Requirement Engineering and Risk Assesment Processes”. Submitted to CAISE 2012.

[2] Minh Sang Tran, Fabio Massacci. “Dealing with Known Unknowns: A Goal-based Approach for Understanding Complex Systems Evolution. Submitted to Software and System Modeling, 2012.

[3] Gábor Bergmann, István Ráth, Gergely Varró, Dániel Varró. “*Change-Driven Model Transformations. Change (in) the Rule to Rule the Change*”. In Software and System Modeling, 2011.

[4] Ábel Hegedüs, Ákos Horváth, István Ráth, Moises C. Branco, Dániel Varró. “Quick fix generation for DSMLs” In IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2011, Pittsbourgh, PA, USA, IEEE Computer Society, 18-22 September 2011.

[5] Virginia N. L. Franqueira, Thein Than Tun, Yijun Yu, Roel Wieringa, Bashar Nuseibeh, “Risk and argument: A risk-based argumentation method for practical security” In 19th IEEE International Requirements Engineering Conference, 239-248, IEEE, 2011.

[6] Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, Bashar Nuseibeh, “Security Requirements Engineering: A Framework for Representation and Analysis” IEEE Trans. Software Eng. 34(1): 133-153, 2008.

[7] Yijun Yu, Thein Than Tun, Alessandra Tedeschi, Virginia N. L. Franqueira, Bashar Nuseibeh. “OpenArgue: Supporting argumentation to evolve secure software systems”. In 19th IEEE International Requirements Engineering Conference, 351-352, IEEE, 2011.

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

[9] Eclipse Modeling Framework (EMF)http://www.eclipse.org/modeling/emf/.

[10] Eclipse Graphical Modeling Framework (GMF)http://en.wikipedia.org/wiki/Graphical_Modeling_Framework.

[11] EBIOS. http://www.ssi.gouv.fr/en/confidence/ebiospresentation.html

[12] CORAS, http://coras.sourceforge.net/, SINTEF.

[13] CRAMM, http://www.cramm.com, Siemens.

[14] OCTAVE, http://www.cert.org/octave/, Carnegie Mellon.

[15] BSIMM, Building Security In Maturity Model, http://bsimm.com/.

[16] ISO/IEC 15288, Systems and software engineering — System life cycle processes, ISO, 2008.

[17] ISO/IEC 12207, Systems and software engineering — Software life cycle processes, ISO, 2008.

[18] ISO/IEC FCD 42010, Architecture description, draft.

[19] V. Normand, E. Félix, “Toward model-based security engineering: developing a security analysis DSML”, ECMDA-FA, 2009

APPENDIX A

Managing Changes with Legacy Security Engineering Processes

Edith Felix, Olivier Delande
Thales
Palaiseau, France
{edith.felix,olivier.delande}@thalesgroup.com

Fabio Massacci, Federica Paci
Department of Information Engineering and Computer Science
University of Trento
Povo, Trento
{Fabio.Massacci,Federica.Paci}@unitn.it

Abstract— **Managing changes in Security Engineering is a difficult task: the analyst must keep the consistency between security knowledge such as assets, attacks and treatments to stakeholders' goals and security requirements. Research-wise the usual solution is an integrated methodology in which risk, security requirements and architectural solutions are addressed within the same tooling environment and changes can be easily propagated.**

This solution cannot work in practice as the steps of security engineering process requires to use artefacts (documents, models, data bases) and manipulate tools that are disjoint and cannot be fully integrated for a variety of reasons (separate engineering domains, outsourcing, confidentiality, etc.). We call such processes *legacy security engineering processes.*

In this paper, we propose a change management framework for legacy security engineering processes. The key idea is to separate concerns between the requirements, risk and architectural domains while keeping an orchestrated view (as opposed to an integrated view). We identify some mapping concepts among the domains so that little knowledge is required from the requirement manager about the other domains, and similarly for security risk manager and the system designer: they can stick to their well known (and possibly certified) internal process. This minimal set of concepts is the *interface* between the legacy processes. The processes are then orchestrated in the sense that when a change affects a concept of the interface, the change is propagated to the other domain.

We illustrate this example by using the risk modeling language (Security DSML) from Thales Research and the security requirement language (SI*) from the Univ. of Trento.

System and software engineering life cycle, Security engineering, Security risks, Requirements, Tooling

INTRODUCTION

Change management in security engineering is a particularly daunting task not only because of the inherently difficulty of the task but also for two concerning factors that characterize modern production in an industrial environment.

System and software engineering in industry is a complex process that is subject to many standard and certification processes, in particular when the critical security infrastructures is at stake.

The need to show compliance with standards e.g ISO 15288 and ISO 12207, respectively for system and software engineering makes often the engineering process quite rigid.

Such rigidity is further increased when those security aspects must be further taken into account. Security standards or best practices must be considered such as ISO 27000, EBIOS,CORAS, CRAMM, OCTAVE, BSIMM [16-20]. The design process must also be compliant with those standards.

For complex systems the security engineering process is also inevitably supported by artefacts (UML models of the system to be, DOORS format for requirements [9], UML risk profiles in CORAS [17] etc), and large companies tend to adapt and customize these artefacts to fit their needs and application domains [14,15]. The combination of these two factors makes each step of security engineering process highly customized and highly rigid and de facto unchangeable, as the switching cost would be too high. We end up with the combination of *legacy software engineering processes.*

So what happens when a security requirement or a threat model changes? For example, in the air traffic management domain, 9/11 has dramatically changed the threat model and implied a different design of the "interface" between cabin and cockpit. Changes must percolate through these structures and they might not get through completely. The solutions proposed by most researchers is to have a unique process integrating security requirements, risk assessment and security architectures [1,2,3].

Contribution of the paper

In this paper we propose a security engineering process where the presence of proprietary steps is not a liability. We focus our attention on the interactions between the *security risk manager*, the *requirement manager*, and the *system designer* and we show how the activities performed by these stakeholders can be orchestrated. The key feature of the orchestrated process is *separation of concern principle*. An important advantage of separation of concern is that in-depth expertise in the respective domains is not a prerequisite. The

orchestrated process allows the separate domains to leverage on each other without the need of full integration. As a counterpart, consistency of concerns should be ensured. We assume that security risk manager, the requirement manager, and the system designer share a minimal set of concepts which is the interface between their respective processes: each process is conducted separately and only when a change affects a concept of the interface, the change is propagated to the other domain.

The paper is structured as follows. Section II introduces the running example based on the evolution of ATM systems that is taking place as planned by the Single European Sky ATM Research (SESAR) Initiative. In Section III we instantiate the requirement and the security and system domains with SI* and Security DSML modeling languages respectively. In Section IV we present the interface between the system engineering process and the risk analysis process. In Section V, we outline the importance of including risk analysis into system engineering process and we illustrate a security engineering process based on the collaboration between the security risk manager, the requirement manager, and the system designer. In section VI, we illustrate the orchestrated process based on the running example in Section II. Section VII presents related works. Section VIII concludes the paper.

II. RUNNING EXAMPLE

To illustrate the change propagation process, we will focus on the ongoing evolution of Air Traffic Management (ATM) systems planned by the ATM 2000+ Strategic Agenda and the Single European Sky ATM Research (SESAR) Initiative [4].

Part of ATM system's evolution process is the introduction of a new decision support tool for air traffic controllers (ATCOs) called Arrival Manager (AMAN) in order to support higher traffic loads. The main goal of the AMAN is to help ATCOs to manage and better organize the air traffic flow in the approach phase. The introduction of the AMAN requires new operational procedures and functions and imposes new security properties to be satisfied. Before the addition of the AMAN, the Sector Team[1] had to manually perform the operations related to the approach phase: the generation of the arrival sequence and the allocation of runaways. Now, some of the operations that were manually done by Sector Teams are performed by the AMAN such as providing sequencing and metering capabilities for runways, airports or constraint points, creating an arrival sequence using 'ad hoc' criteria, managing and modifying proposed sequences, supporting runway allocation at airports with multiple runway configurations, and generating advisories for example on the time to lose or gain, or on the aircraft speed. The introduction of the AMAN requires also the addition of a new role between ATCOs, called Sequence Manager (SQM), who will monitor and modify the sequences generated by the AMAN and will provide information and updates to the Sector Team.

[1] The sector team consists of a Tactical Controller and a Planning Controller.

III. BACKGROUND

We instantiate the requirement framework to SI* [3] and the risk framework to Security DSML [13].

SI* is a requirement framework which supports both early and late requirement analysis. SI* has several extensions, but in this paper we focus on the trust and risk extension proposed in [2]. We only consider a subset of SI* relations, namely *AND/OR decomposition*, *means-end*, *require*, *request*, and *dependency* and *trust relations*. We also consider the *business object* [5] concept which is a combination of goals, processes, and resources.

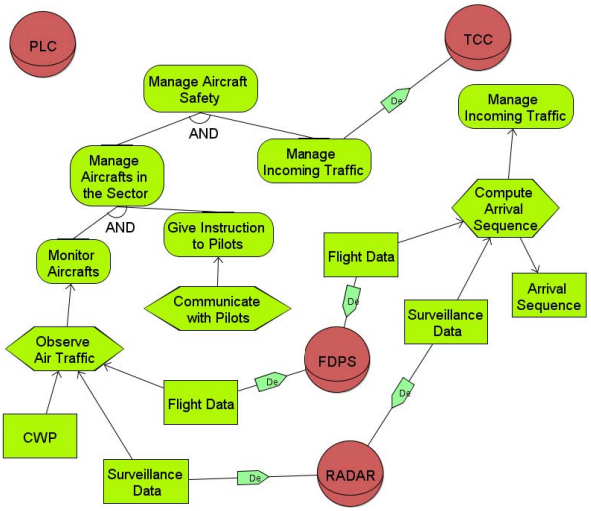


Figure 1. Example of SI* model

The requirement analysis consists of five steps: 1)Identify relevant stakeholders, modeled as *actor* (circle) and its structure; 2) Capture and refine actor's *goals* (rounded rectangle); 3) Define means - i.e., *process* (hexagon) or *resource* (rectangle) - to achieve their goals; 4) Model strategic dependencies between actors in fulfilling/executing/providing some goals/processes/resources; 5) Model specific aspects such as security or risk:
e.g introduce *security goals*, which are goals concerning the fulfillment of security properties [1] or assess the achievement level of high-level goals, such as risk level [2].

The requirement analysis is an iterative process that aims at refining the stakeholders' goals until all goals are achieved.
The results of the analysis process are captured by a SI* model as the one in Figure 1 illustrating the running example introduced in Section 2. The model consists of four actors: Planning Controller (PLC), Tactical Controller (TCC), Radar and Flight Data Processor (FDPS). The PLC has one main goal that is Manage Aircraft Safety that is decomposed into Manage Aircraft in the Sector and Manage Incoming Traffic subgoals. The latter goal is delegated to TCC who fulfills by providing the process

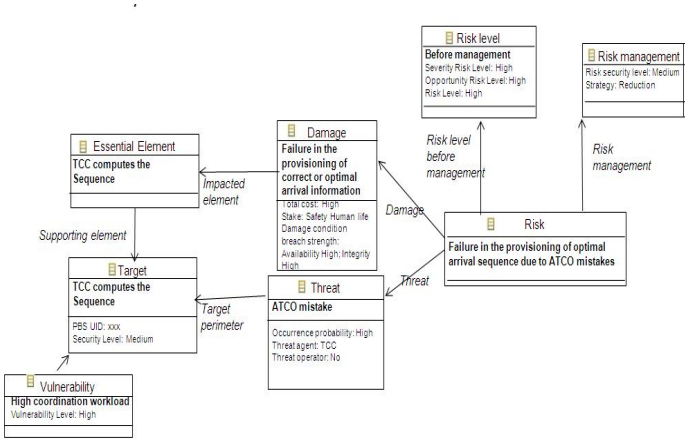


Figure 2. An example of Security DSML model

Compute Arrival Sequence. This task requires the resources Flight Data and Surveillance Data that are provided by the FDPS and the Radar respectively.

Security DSML is the language and a tool developed to capture the security risk analysis concepts derived from the French EBIOS methodology [16]. As a tool, Security DSML realizes a Viewpoint of a system Architecture Model as defined in coming ISO 42010 standard [23].

The main security concepts are the following:

- *Essential element*: an element of the system at Business Architecture or Service-oriented Architecture Plans.
- *Damage*: the impact related to a risk on the essential elements of system.
- *Target*: an element of the system potentially threatened by one or more threats.
- *Vulnerability*: weakness in a system, system security procedures, internal controls, or implementation that could be exploited.
- *Threat*: any circumstance or event with the potential to adversely impact a system through unauthorized access, destruction, disclosure, modification of data, and/or denial of service.
- *Risk*: possibility that a particular threat will adversely damage an element of the system design.
- *Security objective*: expression of the intention to counter identified risks by goals regarding the security of the system.
- *Security requirement*: a functional or assurance general specification covering one or more security objectives.
- *Security solution*: a security measure that implements a security requirement.

Figure 2 shows the ATM example. The model starts from the activity TCC computes the sequence called an *Essential Element* in Security DSML language. A potential *Damage* Failure in the provisioning of correct or optimal arrival information is identified. The ATCO[2] as supporting elements of the activity, called *Targets* in Security DSML, are vulnerable to High coordination workload, and are subject to the *Threat* ATCO mistake. Then the *Risk* Failure in the provisioning of correct or optimal arrival information is identified, which has a high risk level, which needs to be reduced to at least medium.

IV. CONCEPUAL MAPPING

Even though conducted separately, the requirement analysis, and the risk analysis processes can be orchestrated so that they can benefit from the respective results. In order to allow the orchestration between these processes, we need to identify a set of concepts that is the *interface* between them (see Table I).

TABLE I. INTERFACE

	Conceptual Mapping		
Requirement	***Risk***	***Architecture***	***Type***
Business Object	Essential Element		Shared
Goal	Security Objective		Mapped
Security Goal	Security Requirement		Mapped
Process		Security Solution	Mapped

We distinguish the interface concepts in *shared elements* and *mappable elements*. The shared elements are model elements that conceptually have the same semantic in the three domains. The mappable elements are elements from one domain that are not shared by the other, but nevertheless can be mapped to elements of the other domain.

When a change affects a mappable or shared element in one domain such change is propagated to the other domain. The following table summarizes the conceptual mapping.

V. CHANGE MANAGEMENT PROCESS

International standards like ISO/IEC 15288:2008 and ISO/IEC 12207:2008 [21, 22] describe the system and software life cycle of the engineering process and including clauses mentioning that non-functional properties such as security should be considered in different phases.

In security specialty engineering, risk analysis methodologies such as EBIOS, CORAS or CRAMM serve security risk managers to produce a rationale for security requirements and assess the risks in an exhaustive way, as needed in domains such as administration or military systems. The risk management process does not cover the entire security engineering activities but is a key starting point to them.

Thus a first issue is to show how the risk management process and security requirement analysis can collaborate with the global system engineering process described in those

[2] ATCO stands for Air Traffic Controller, which here means Planning Controller (PLC) and Tactical Controller (TCC).

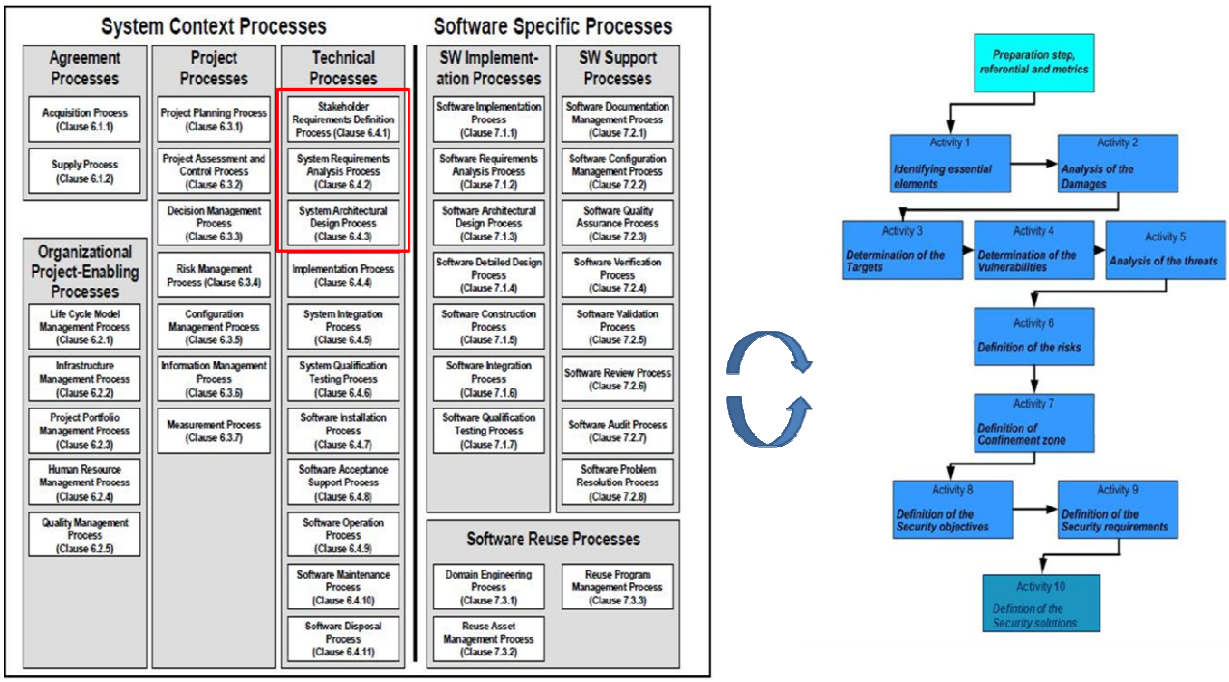


Figura 3. ISO/IEC 12207 vs EBIOS process

engineering standards. The difficulty resides in the necessary iterations needed to refine the security requirements since some vulnerabilities and risks will appear only once the system architectural design has been set up.

This article focuses on how the risk engineering process which has been standardized independently can be orchestrated into the overall system engineering process. In particular we investigate how the processes Stakeholder Requirements Definition (Clause 6.4.1), Requirements Analysis (Clause 6.4.2), and Architectural Design (Clause 6.4.3) processes of ISO/IEC 12207 can be orchestrated with EBIOS risk methodology activities (see Figure 3).

The resulting orchestrated process is represented in Figure 4. The stakeholder relationship[3] technical manager gets the needs and requirements from the stakeholders (Clause 6.4.1). He pushes the information related to security needs to the security risk manager who expresses the unwanted damages and defines the first security objectives. The stakeholder relationship technical manager validates the security objectives with the stakeholders and consolidates them before sending them to the requirement manager. Then the requirement manager consolidates them with requirements from other stakeholders and sends all the requirements to the system designer.

The system designer analyzes the requirements (Clause 6.4.2) and defines the functions of the system. Once this is done, the security risk manager updates the essential elements (Activity 1) based on the functions of the system, updates the damages (Activity 2), adds some security objectives (Activity 8), defines first security requirements (Activity 9) and sends them to the system designer and to the requirement manager. The system designer validates the requirements analysis with the design authority who propagates it.

The system designer proceeds to architectural design of the system, allocating functions to elements of the system (Clause 6.4.3). This new organization of the model of the system is analyzed by the security risk manager who evaluates carefully the risks and defines security solutions (Activities 3 to 10), and sends the updates of the security solutions to the system designer who consolidates the system design and validates the architectural design with the design authority who propagates it. Architectural design and updated requirements are propagated to the system engineering manager and the security engineering manager for them to complete the design at physical layer and implement it (Clause 6.4.3 and Clause 6.4.4).Once he has chosen the security solutions, the security engineering manager sends the information to the security risk manager for targets and vulnerabilities determination (Activities 3 and 4) and a full update of the risk management cycle (Activities 5 to 10).

The updates are passed through the security engineering manager to the system engineering manager. The system engineering manager validates the architectural design and the existing elements of the implementation with the design authority.

VI. APPLICATION TO THE ATM DOMAIN

Here we illustrate some of the steps of the integrated process that involves the security risk manager, the requirement engineer and the system designer, by using the evolution scenario introduced in Section II.

1) The stakeholder relationship technical manager and the security risk manager interact to identify an initial set of security objectives to be passed to the requirement manager.

[3] The stakeholder relationship manager in the majority of requirement engineering framework is called requirement manager

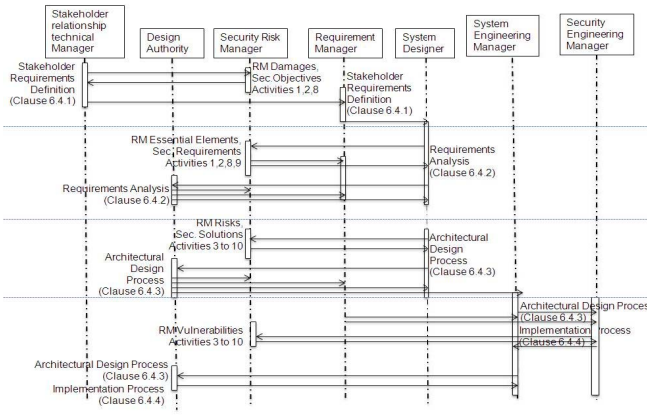


Figure 4.ISO/IEC 12207 sub-set of processes and EBIOS collaboration diagram

2) The stakeholder relationship technical manager passes the requirements proposed by the stakeholders and the initial security objectives to the requirement manager. A change request is triggered for the requirement domain: the SI* model illustrated in Figure 1 is produced by the requirement manager.

3) The system designer analyzes the SI* model provided by the requirement manager and then passes it to the security risk manager.

4) The security risk manager identifies the following new security objectives:
   - **O1** The system shall be computed automatically by an Arrival Manager system that covers the risk
     - **R1** Failure in the provisioning of correct or optimal arrival information due to ATCO mistakes.
   - **O2** The update of the system should be handled through a dedicated role of Sequence Manager that covers the risk **R1**.

   The above security objectives are refined into the following security requirements:
   - **RE1** The system should integrate an AMAN (refines security objective **O1**)
   - **RE2** The organization should integrate a SQM (refines security objective **O2**).

   Figure 5 represents the Security DSML model updated with the new security objectives and security requirements.

5) The changes into the Security DSML model trigger a change request for the requirement domain. The requirement manager receives the new security objectives and requirements and updates the SI* model as shown in Figure 6: two new actors, AMAN and the SQM have been added with their goals, process and resources.

6) The new processes Compute Arrival Sequence provided by AMAN and Monitor and Modify provided by SQM identified by the requirement manager has to be propagated to the system designer and to the security risk manager.

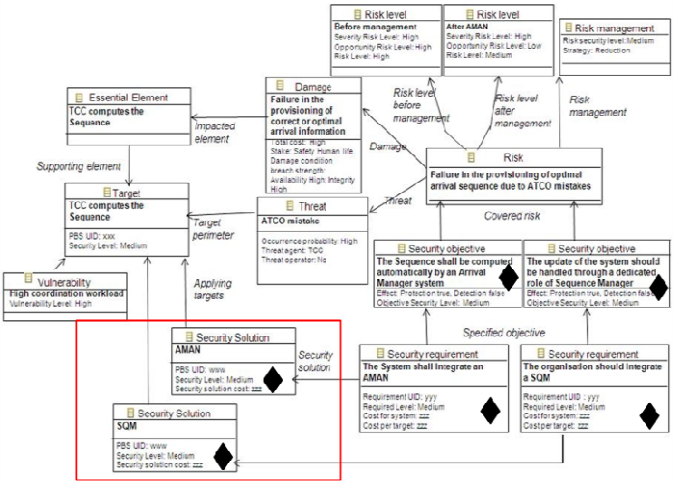


Figure 5. Security DSML Model after the introduction of AMAN

The security risk manager assesses the new processes proposed by the requirement manager and defines new security solutions to match the processes (outlined in red in Figure 5). Then, the security risk manager passes the identified security solutions to the system designer for validation.

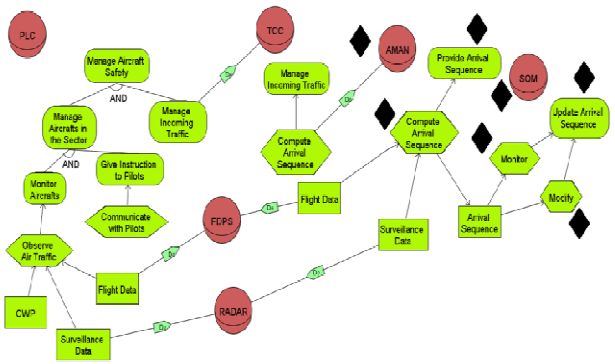


Figure 6. SI* Model after the introduction of the AMAN and SQM

VII. RELATED WORK

The predominant standards for system and software engineering are ISO/IEC 15288 and 12207 [21, 22]. Upcoming ISO/IEC 42010 [23] standard describes the common vocabulary and framework for working on several concerns and specialty engineering viewpoints which all refer to a common architecture description.

Among the security risk analysis methods CORAS [17] is based on UML environment and has proposed new techniques for structured diagrams. EBIOS [16] released a simplified EBIOS 2010 methodology which is more suitable for architectural engineering environment.

Existing requirement engineering proposals have been extended to include security concepts in the requirement

conceptual models and to support security related analysis. Van Lamswerde extended KAOS [1] by introducing the notion of obstacles to capture exceptional behaviours and anti-goals to model the intention of an attacker to threaten security goals. Massacci et al.[3] have defined Secure Tropos for modeling and analyzing authorization, trust and privacy concerns. Haley et al. [6] extend problem frames to determine security requirements for the system by considering possible security threats. Elahi et al. [7] extend i* with security related notions (e.g., attacker, vulnerability, malicious goal) for capturing and analyzing the impact of vulnerabilities to software systems. Asnar et al. [2] extend also i* with the notion of uncertain events and treatments to support the risk assessment process into requirement engineering process.

With respect to these proposals, our work does not require to extend existing requirement frameworks at modeling and process level but it just requires the requirement analyst and the risk analyst to share the understanding of a set of concepts to be able to communicate and share the results of the respective analysis processes. Moreover, our orchestrated process has also another advantage: it supports change propagation between the requirement and risk domain which is enabled by the shared interface.

In fact, only some requirement engineering proposals provide support for handling change propagation and for change impact analysis.

Chechik et al. [11] propose a model-based approach to propagate changes between requirements and design models that utilize the relationship between the models to automatically propagate changes. Hassine et al. [12] present an approach to change impact analysis that applies both slicing and dependency analysis at the Use Case Map specification level to identify the potential impact of requirement changes on the overall system. Lin et al. [10] propose capturing requirement changes as a series of atomic changes in specifications and using algorithms to relate changes in requirements to corresponding changes in specifications.

VIII. CONCLUSIONS

We have proposed a change management framework for legacy security engineering processes. The key idea is to separate concerns between the requirements, risk and architectural domains while keeping an orchestrated view. The orchestration has been based on the mapping of concepts among the domains so that little knowledge is required from the requirement manager about the other domains, and similarly for security risk manager and the system designer. The processes are then orchestrated in the sense that when a change affects a concept of the interface, the change is propagated to the other domain.

We have illustrated the framework using an example of evolution taken from the air traffic management domain. We are planning to apply the framework to other industrial case studies e.g the evolution of multi-application smart cards.

ACKNOWLEDGMENT

This work has been partly funded by EU project - Network of Excellence on Engineering Secure Future Internet Software (NESSoS) and by the EU-FP7-FET-IP-SecureChange project.

REFERENCES

1. A. van Lamsweerde, “Elaborating security requirements by construction of intentional anti-models,” in Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on, 2004, pp. 148–157.
2. Y. Asnar, P. Giorgini, and J. Mylopoulos, “Goal-driven risk assessment in requirements engineering,” Requirements Engineering, pp. 1–16, (to appear).
3. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, “Requirements engineering for trust management: model, methodology, and reasoning,” International Journal of Information Security, vol. 5, no. 4, pp. 257–274, Oct. 2006.
4. EUROCONTROL, “ATM Strategy for the Years 2000+,” 2003.
5. Y. Asnar, P. Giorgini, P. Ciancarini, R. Moretti, M. Sebastianis, and N. Zannone, “Evaluation of business solutions in manufacturing enterprises,” International Journal on Business Intelligence and Data Mining, vol. 3, no. 3, pp. 305 – 329, 2008.
6. C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, “Security requirements engineering: A framework for representation and analysis,” IEEE Trans. Softw. Eng., vol. 34, pp. 133–153, January 2008.
7. G. Elahi, E. Yu, and N. Zannone, “A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities,” Requirements Engineering, vol. 15, pp. 41–62, 2010.
8. L. Liu, E. Yu, and J. Mylopoulos, “Security and privacy requirements analysis within a social setting,” in Proceedings of the 11th IEEE International Conference on Requirements Engineering. Washington, DC, USA: IEEE Computer Society, 2003, pp. 151–161.
9. DOORS. http://www-01.ibm.com/software/awdtools/doors/.
10. L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, “Introducing abuse frames for analysing security requirements,” in Proceedings of the 11th IEEE International Conference on Requirements Engineering. Washington, DC, USA: IEEE Computer Society, 2003, pp. 371–372.
11. M. Chechik, W. Lai, S. Nejati, J. Cabot, Z. Diskin, S. Easterbrook, M. Sabetzadeh, and R. Salay, “Relationship-based change propagation: A case study,” in Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering, ser. MISE ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 7–12.
12. J. Hassine, J. Rilling, and J. Hewitt, “Change impact analysis for requirement evolution using use case maps,” in Proceedings of the Eighth International Workshop on Principles of Software Evolution. Washington, DC, USA: IEEE Computer Society, 2005, pp. 81–90.
13. V. Normand, E. Félix, “Toward model-based security engineering: developing a security analysis DSML”, ECMDA-FA, 2009.
14. Eclipse Modeling Framework (EMF)http://www.eclipse.org/modeling/emf/.
15. Eclipse Graphical Modeling Framework (GMF)http://en.wikipedia.org/wiki/Graphical_Modeling_Framework.
16. EBIOS. http://www.ssi.gouv.fr/en/confidence/ebiospresentation.html
17. CORAS, http://coras.sourceforge.net/, SINTEF.
18. CRAMM, http://www.cramm.com, Siemens.
19. OCTAVE, http://www.cert.org/octave/, Carnegie Mellon.
20. BSIMM, Building Security In Maturity Model, http://bsimm.com/.
21. ISO/IEC 15288, Systems and software engineering — System life cycle processes, ISO, 2008.
22. ISO/IEC 12207, Systems and software engineering — Software life cycle processes, ISO, 2008.
23. ISO/IEC FCD 42010, Architecture description, draft.

APPENDIX B

A Change-Driven Interplay between Requirement Engineering and Risk Assessment Processes

Yudistira Asnar[1], Fabio Massacci[1], Federica Paci[1], and Bjørnar Solhaug[2]

[1] DISI - University of Trento,
{yudis.asnar, fabio.massacci,federica.paci}@unitn.it
[2] SINTEF ICT, Norway
bjornar.solhaug@sintef.no

Abstract. Maintaining links between security knowledge and requirements is crucial to analyze the impact of requirements evolution on the system's security risks and vice versa. Some security requirements engineering proposals aim to create a link with the security risk domain by extending requirement conceptual models and by including security risk analysis as part of the requirement engineering process. These approaches facilitate change propagation, but they do not correspond to the practice in industrial software/system engineering processes: risk assessment and requirement elicitation activities are usually conducted separately. This paper proposes an approach for change propagation between requirement engineering and risk assessment activities that is based on orchestration rather than on integration. The orchestration relies upon mappings between key concepts of the requirement and the risk conceptual models. Key aspects of the approach are *separation of concern* and *bidirectional consistency* of requirement and risk models.

1 Introduction

Changing requirements might give rise to potential security risks that in turn require some treatments to ensure and maintain an acceptable security risk level. Or treatment options that result from risk assessment may lead to new security requirements that should be included in the requirement model. Moreover, the requirement changes may involve new assets the risk level of which needs to be assessed. Thus, there is the need to trace changes to security knowledge such as assets, attacks and treatments to stakeholders' goals and security requirements and vice versa.

The problem of creating a "link" between risk and requirements is important and has been investigated in several research works [2, 15, 23–25]. These frameworks have addressed the problem by extending requirements conceptual models with security related concepts and by incorporating risk analysis into the requirement analysis. These approaches work well in theory because they facilitate change propagation. However, these approaches do not reflect the common practice in industrial software/system engineering processes. Software/system engineering processes are supported by artifacts (documents, models, data bases) that are disjoint and cannot be fully integrated for a variety of reasons (separate engineering domains, outsourcing, confidentiality, etc.). Thus, the collaboration between risk analyst and requirement analyst in such processes

is sometimes difficult, especially when a change occurs and they have to interact to keep the risk and the requirement models mutually consistent under change.

This paper proposes an alternative solution to change propagation that is not based on the integration of requirement engineering and risk assessment in a unique methodology, but on the *orchestration* of the two. The orchestration relies upon mappings between key concepts of the requirement and the risk conceptual models. The requirement engineering process and the risk analysis process are orchestrated when a change is applied to model elements that are instances of the mapped concepts. Orchestration has several advantages with respect to integration. The first one is that orchestration supports *separation of concern*: the requirement analyst and the risk analyst do not need to have in-depth expertise in the respective domains, they only need to know the mapped concepts on which the orchestration is based. The second advantage is that the orchestration ensures consistency between the requirement and the risk models and that neither of them becomes obsolete with respect to the other under change.

We have expressed the mappings between concepts of requirement and risk conceptual models as VIATRA2 [29] graph transformation rules. VIATRA2 ensures the automatic creation of traceability links between requirement and risk models and the automatic execution of the mappings when a change affects a mapped concept.

To illustrate the change-driven interplay between requirement engineering and risk assessment, we will focus on the ongoing evolution of ATM systems as planned by the ATM 2000+ Strategic Agenda [9] and the SESAR Initiative.[3] The current evolution of ATM systems provides us concrete examples of changes in the requirements domain that have an impact on the risk domain and vice versa. As part of the ATM system's evolution, we consider the introduction of a new surveillance tool, the Automatic Dependent Surveillance-Broadcasting (ADS-B). ADS-B provides accurate aircraft position information by using a global navigation satellite system. If on one side ADS-B is beneficial for ATM, on the other side it makes ATM systems vulnerable to new threats. In fact, ADS-B transmissions can be easily corrupted: a concrete example is the spoofing of the GPS satellite that provides the GPS signal to determine aircraft position.

The structure of the paper is as follows. In Section 2, we give an overview of SI* [2, 10, 22] and of CORAS [21] as examples of instantiations of the requirement and the risk domains, respectively. In Section 3 we define the conceptual mappings that are the basis for the change-driven interplay. In Section 4, the change-driven interplay is presented and illustrated with concrete examples. We conclude the paper with the related works in Section 5 and outlining future work in Section 6.

2 Background

For sake of concreteness, we instantiate the requirement framework to SI* and the risk framework to CORAS. However, our approach is independent from the specific requirement and risk frameworks that are adopted, and can thus be applied to other competing instantiations if these have concepts similar to the ones we propose in Section 3.

[3] `http://www.sesarju.eu/`

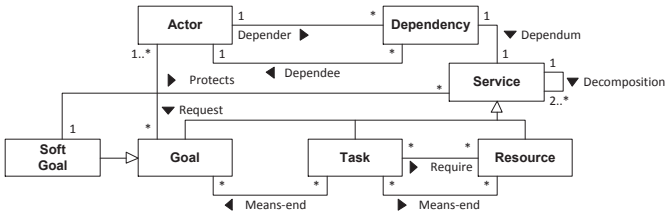


Fig. 1. SI* conceptual model

2.1 SI* Framework

SI* is a modeling framework extending the i* framework [30] to support security requirement analysis. SI* aims at analyzing and modeling organizational settings and its security and dependability requirements. The main concepts of the SI* language[4] are represented in Figure 1: an *actor* is an entity which has intentions, capabilities, and entitlements; a *goal* captures a strategic interest that actor intends to see fulfilled; a *soft goal* captures a non-functional requirement; a *resource* is an artifact produced/consumed by a goal; *AND/OR decomposition* is used to refine a goal; *means-end* identifies goals that provide means for achieving another goal or resources produced or consumed by a goal/task. SI* also captures social relationships (e.g., *delegation* and *trust*) for defining the entitlements, capabilities and objectives of actors. A *delegation* marks a formal passage of responsibility (*delegation execution*) or authority (*delegation permission*) from an actor (*delegator*) to the actor receiving the responsibility/authority (*delegatee*) to achieve a goal or to provide a resource. *Trust* is a relation between two actors representing the expectation of one actor (*trustor*) about the capabilities of the other (*trustee*) – *trust execution*, and about the behavior of the trustee with respect to the given permission – *trust permission*.

To specify security needs which identify assets that have to be protected by preventing loss of confidentiality, integrity, availability, authenticity or accountability, we use the concept of *soft goal* and we introduce a new relation *protects* between a soft goal and a *service* – a resource, task or goal – to denote that the service is an asset.

The requirement analysis supported by SI* is an iterative process that aims at refining the actors' goals until all high-level goals are achieved. We consider the requirement analysis proposed in [2] where the keyword SAT denotes that the evidence is in favor of the achievement of a goal, and DEN to denote that the evidence is against it. The results of the analysis are captured by a SI* model that illustrates the detailed goals that serve stakeholders' needs, how they are achieved, the necessary resources, and the security goals that protect the system.

Example 1. Figure 2 (a) shows an example of SI* model that describes some actors involved in aircrafts arrival procedures. There are four main actors: the Tactical Controller (TC), the FDPS (flight data processing system), the RDPS (radar data processing system), and the Radar. The TC has a main goal, namely Monitor air traffic which needs

[4] We only mention the concepts that are relevant to this work

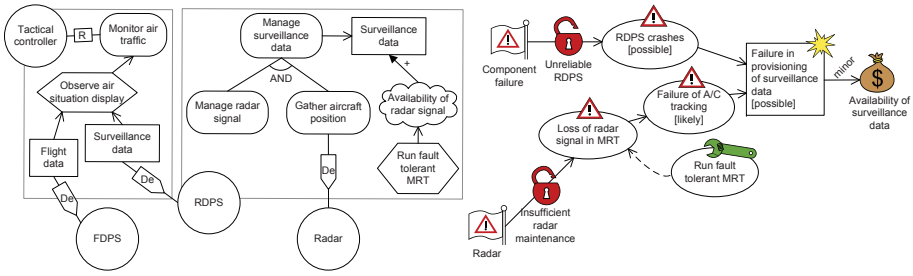


Fig. 2. SI* and CORAS models examples

surveillance data to be fulfilled. The provision of Surveillance data is delegated to the RDPS. The RDPS is concerned about the availability of the radar signal as expressed by the soft goal Availability of radar signal since the radar signal is crucial for the computation of aircrafts' arrival sequence.

2.2 CORAS

Risk management may be referred to as activities to direct and control an organization with regard to risk. CORAS is a model-driven approach to risk management that is based on the ISO 31000 standard for risk management [13], and offers a method for risk analysis. The CORAS risk analysis method is supported by the CORAS language, that serves as the basis for building different kinds of diagrams for risk modeling and analysis. The method and language are firmly based on a set of well-defined risk related concepts. The most important of these, as well as the relationships between them, are shown in the UML class diagram of Figure 3.

An *unwanted incident* is an event that harms or reduces the value of an asset, where an *asset* is something to which a party assigns value and hence for which the party requires protection. A *party* is a stakeholder, i.e. an organization, company, person, group or other body, on whose behalf a risk analysis is conducted. A *risk* is the likelihood of an unwanted incident and its consequence for a specific asset, where the *likelihood* is the frequency or probability for something to occur and the *consequence* is the impact of an unwanted incident on an asset in terms of harm or reduced asset value. A *threat* is the potential cause of an unwanted incident, whereas a *threat scenario* is a chain or series of events that is initiated by a threat and that may lead to an unwanted incident. A *vulnerability* is 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. Finally, a *treatment* is an appropriate measure to reduce risk level.

Example 2. The CORAS threat diagram of Figure 2 (b) documents a risk with respect to the asset Availability of surveillance data. The provisioning of surveillance data partly relies on the radar data processing system (RDPS) and aircraft (A/C) tracking by radar, and failure in any of the two may lead to loss of availability.

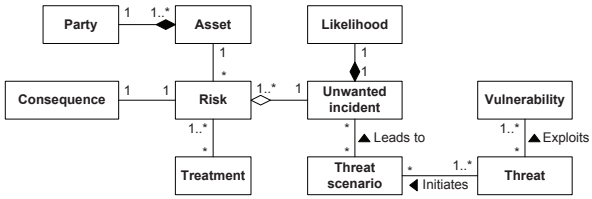


Fig. 3. CORAS conceptual model

The CORAS risk analysis consists of five main phases. The *context establishment* includes defining the target of analysis, setting the scope and focus of the analysis, and deciding the risk evaluation criteria. The *risk identification* is to identify, model and document risks, including the various sources of risk such as threats, vulnerabilities and unwanted incidents. The *risk estimation* is to determine the risk levels by estimating the consequences of unwanted incidents and their likelihood to occur. The *risk evaluation* is to compare the risk estimation results with the risk evaluation criteria and decide which risks need to be evaluated for treatment. The *risk treatment* is to identify strategies for mitigating risk, reducing them to an acceptable level. The process is typically iterative and aims at protecting stakeholders' assets, and terminates when the residual risk is equal to or lower than the risk tolerance.

3 Conceptual Mappings

SI* and CORAS models are rarely static as they may continuously evolve during their respective processes. Evolution is driven by user input in modeling environments and editors. Two types of changes can be applied to a model: *atomic* changes which are model updates consisting of an operation – e.g create, delete, update – and operands (model elements), or a complex *sequence* of such atomic operations. Thus, a change in a SI* or CORAS model can be represented as a transition of the model from a *pre-state*, to a *post-state*. The difference between the pre-state and the post-state model is called *change delta* (or model delta). Model manipulation operations are specified in a *change request* that can be proposed either by the requirement or the risk analyst.

Since SI* and CORAS conceptual models evolve independently, inconsistencies between the two models can be identified only when a periodic review is conducted. Thus, to ensure bidirectional consistency, i.e. maintain mutual consistency under change, we propose a change-driven interplay between the requirement analyst and the risk analyst. The interplay is based on a set of mappings between key concepts of the SI* and CORAS conceptual models. We have mapped concepts 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. We have identified three conceptual mappings: **ServiceToAsset**, **ServiceToTreatment**, and **TreatmentToTask**.

ServiceToAsset. A *service* that is linked to a *soft goal* by a *protects* relation denotes a resource, a task or a goal that is of value for the organization and thus needs to be

protected from harm. Since in CORAS, an asset is something of value that requires protection, we map a *service protected by a soft goal* in SI* to an *asset* in CORAS.

ServiceToTreatment. A *service* which is related to another service mapped to an asset in a CORAS model, can be mapped to a *treatment* if the service reduces the likelihood or the consequence of a threat scenario damaging the asset.

TreatmentToTask. A *treatment* is a security control that should reduce the likelihood or consequence of a threat to an asset which results in a loss of confidentiality, availability, or integrity. Thus, if a treatment is implemented, the confidentiality, integrity or availability of an asset is protected. A *treatment* in CORAS can therefore be mapped to a *task* which fulfills the *soft goal* which specifies the security property that has to be preserved for an asset.

We have formalized the mappings in the VIATRA2 graph transformation language [29]. The VIATRA2 framework supports model-to-model transformations: a model transformation is a program that receives as input a source model that conforms to its source metamodel and produces a target model conforming to a target metamodel. The VIATRA2 transformation language consists of several constructs: *graph patterns* are declarative queries, that specifies constraints and conditions on models; *graph transformations rules* (GT) provide a declarative, high-level rule- and pattern-based manipulation language for models; abstract state machine (ASM) rules can be used to assemble patterns and graph transformation rules to specify complex model transformations. A *graph transformation rule* can be specified by using a left-hand side – LHS (or precondition) pattern determining the applicability of the rule, and a right-hand side – RHS (postcondition) pattern which specifies how the model should be changed at the matches of the LHS.

The mappings **ServiceToAsset**, **ServiceToTreatment**, and **TreatmentToTask** are formalized by the graph transformation rules *newAsset*, *potentialTreatment*, and *transformTreatment* respectively. The rules are executed on a model space which includes a) SI* conceptual model, b) CORAS conceptual model, c) a SI* model, d) a CORAS model. The model space includes also the traceability conceptual model which specifies the structure of the traceability matrix and the traceability model storing the mappings between the elements of the CORAS and the SI* models. The traceability links are automatically created by the graph transformation rules.

The *newAsset* rule has three parameters: `SecGoal`, the soft goal that has to be mapped, `CModel`, the CORAS model, and `TraceModel`, the traceability model that stores the mappings between the elements of the SI* and CORAS model. The precondition calls two patterns: `secgoalProtectsAsset` which checks that `SecGoal` is a soft goal that is linked to a service by the ”protects” relation; the negative condition pattern `mappedElement` which checks that soft goal `SecGoal` has not already been mapped to an asset in the CORAS model `CModel`. The postcondtion creates an asset `Asset` in the CORAS model `CModel`. The action of the rule names the asset `Asset` created in `CModel` as `SecGoal` and creates a traceability link in the traceability model who has as source of the mapping `SecGoal` and as target `Asset`.

```
gtrule newAsset(out SecGoal, out CModel, out TraceModel)={
        precondition pattern unmappedsecurityGoal(SecGoal,
            SecGoalName, AssetName)={
            find secgoalProtectsAsset(SecGoal, SecGoalName,
                AssetName);
            neg find mappedElement(SecGoal, TraceModel, Source,
                Target);
        }
        postcondition pattern mappedAsset(Asset, CModel)={
            Asset(Asset) in CModel;
            CDiagram.entities(Rel, CModel, Asset);
        }
        action {
            call copyName(SecGoal, Asset);
            call createLink(SecGoal, Asset, TraceModel);
        }
    }
```

Listing 1.1. Mapping Soft Goal to an Asset

```
        gtrule potentialTreatment(out Service, out CModel, out
            TraceModel) = {
        precondition pattern unmappedTreatment(Service, SecGoal,
             Asset) = {
            find secgoalProtectsAsset(SecGoal, SecGoalName,
                AssetName);
            find ServiceToAsset(Service, SecGoal, Asset);
            neg find mappedElement(Service, TraceModel, Source,
                Target);
        }
        postcondition pattern newPotentialTreatment(SecGoal,
            Asset, TraceModel) = {
            find mappedAsset(Asset, TraceModel);
            }
        action {
            println( name(Service) + " is a potential treatment
                for Asset"+ name(Asset));
        }
    }
```

Listing 1.2. Mapping a new Service to a Potential Treatment

The *potentialTreatment* rule has three parameters: `Service`, the service that has to be mapped, `CModel`, the CORAS model, and `TraceModel`, the traceability model. The precondition calls three patterns: `secgoalProtectsAsset` which looks for a a soft goal `SecGoal` that "protects" a service `Asset`; `ServiceToAsset` checks if the service `Service` is related to the service `Asset` protected by the soft goal `SecGoal`; the negative condition pattern `mappedElement` which checks that the service `Service` has not already been mapped to a treatment in the CORAS Model

CModel. The postcondtion checks the asset in the CORAS model CModel that is mapped to the service Asset protected by the soft goal SecGoal. The action presents Services as a potential treatment for asset Asset.

```
        gtrule transformTreatment(out Treatment, out ReqModel, out
             TraceModel)={
        precondition pattern unmappedTreatment(Treatment, Asset)
            ={
            find harmedAsset(Treatment, ThreatScenario, Asset);
            neg find mappedElement(Treatment, TraceModel, Source,
                Target);
        }

        postcondition pattern newTask(Asset, Task, ReqModel,
            TraceModel)={
            find sistarmodel(ReqModel);
                 find task(Task);
                 find mappedSecGoal(Asset, SecGoal);
                 find meansEnd(R, Task, TaskName, SecGoal,
                     SecGoalName);
            }
        action {
            call copyName(Treatment, Task);
            call createLink(Treatment, Task, TraceModel);
        }
    }
```

Listing 1.3. Mapping Treatment to Task fulfilling a Soft Goal

The *transformTreatment* rule has three parameters: Treatment, the treatment to be mapped, ReqModel, the SI* model, and TraceModel, the traceability model. The precondition specifies two constraints for Treatment: harmedAsset identifies the asset Asset that is harmed by the threat scenario ThreatScenario treated by Treatment; the negative condition pattern mappedElement checks that Treatment has not already been mapped to a task in the SI * model. The postcondtion creates a task Task in the SI* model ReqModel and links it by means of a means-end relation to the soft goal SecGoal that protects the service mapped to Asset. The action of the rule names the asset Task created in ReqModel as the treatment Treatment and creates a traceability link in the traceability model who has as source element of the mapping Treatment and as target Task.

4 Change-driven interplay between risk and requirement analysts

The interaction between the risk analyst and requirement analyst is triggered when a change is applied to model elements that are instances of the mapped concepts in SI* and CORAS conceptual models. In the following we present three *bidirectional propagation* scenarios where changes are propagated from the requirement models to the risk model and vice versa. These are examples of possible scenarios that are supported by the conceptual mappings.

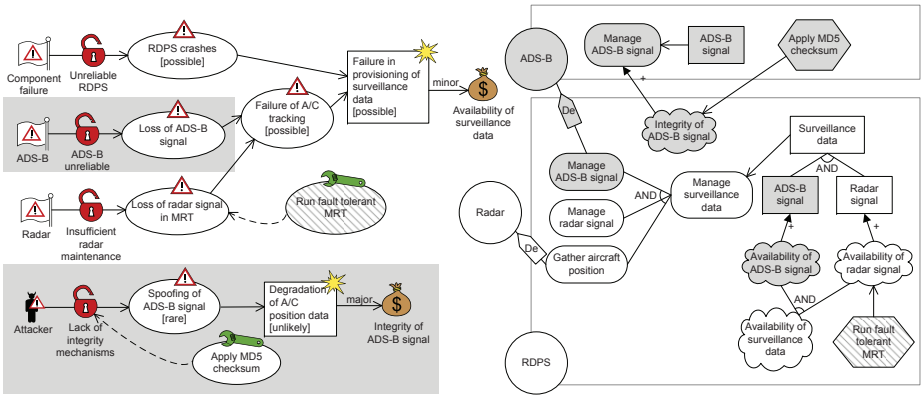


Fig. 4. Change-driven interplay

ServiceToAsset-TreatmentToTask. When a new soft goal and a protected service are added to the requirement model, a new asset is created in the risk model as specified by the graph transformation rule *newAsset* (see Listing 1.1). Hence, the requirement analyst passes the new soft goal and the service protected by the goal to the risk analyst. The risk analyst, then, decides if it is the soft goal, the service or both that are mapped to an asset in the risk model. When the asset is added, the risk analyst identifies possible threat scenarios and estimates the associated risks for the asset. If the level of the identified risks is not acceptable, the risk analyst has to select a set of treatments that reduce the risks to an acceptable level. The identified treatments have to be proposed to the requirement analyst who decides whether to implement them or not based on cost/benefit analysis. Thus, the graph transformation rule *transformTreatment* (see Listing 1.3) is applied to the SI* model, and thus new tasks representing the treatment proposed by the risk analyst is added to the requirement model. The task is linked by a means-end relation to the soft goal and the protected service mapped to the asset which is threatened by the risk mitigated by the treatment. The tasks that correspond to treatments that the requirement analyst decided not to implement, are deleted from the requirement model. Then, the requirement analyst has to return the list of the treatments selected for implementation to the risk analyst. Then, the risk analyst can remove the threat scenarios mitigated by the treatments from the assumptions on which the risk estimates have been calculated.

Example 3. To illustrate the scenario above, we consider as prestate models the SI* and CORAS models illustrated in Figure 2. The change we consider first is the introduction of a new actor, the ADS-B, which is intended to increase the accuracy and availability of the surveillance data. The changes that are handled are highlighted in gray in Figure 4. The ADS-B introduction requires the addition of the goal Manage ADS-B signal and the resource ADS-B signal. Moreover, since the integrity of the ADS-B signal is critical, this security need is specified by introducing a new soft goal Integrity of ADS-B signal. Notice also that because ADS-B signal becomes part of the surveillance data, the soft goal Availability of surveillance data is decomposed into the subgoals Availability

of ADS-B signal and Availability of Radar signal. The introduction of the ADS-B actor may affect the risks due to the introduction of the new soft goals. In particular, the soft goal Integrity of ADS-B signal is mapped to a corresponding asset in the CORAS model for which a risk assessment is conducted. As ADS-B is prone to data spoofing, this issue is addressed and identified as an unacceptable risk with respect to integrity. The treatment Apply MD5 checksum is considered, and leads to the addition of a new task Apply MD5 Checksum fulfilling the soft goal Integrity of ADS-B signal in the SI* model. The new soft goal Availability of ADS-B signal is also mapped to the CORAS model as a new aspect of the more general asset Availability of surveillance data. The risk analyst decides not to decompose this asset with respect to ADS-B and radar when assessing the impact of the ADS-B. A new threat scenario Loss of ADS-B signal is identified, but the overall risks with respect to availability of surveillance data do not increase; rather, the likelihood of the threat scenario Failure of A/C tracking decreases from likely (cf. Figure 2 (b)) to possible.

ServiceToTreatment-TreatmentToTask. The interaction between the requirement analyst and the risk analyst can also be triggered when a new service is added to the requirement model. If the service is related to another service which is mapped to an asset in the risk model, and the former service reduces the likelihood or the consequence of a threat scenario damaging the asset, the service can be considered as a potential treatment. By a service related to an asset we mean a goal or a task that consumes the asset or a resource that is part of the asset. The execution of the graph transformation rule *potentialTreatment* (see Listing 1.2) is triggered and, thus, a potential treatment is suggested to the risk analyst, who evaluates which is the impact of it on the risk profile. The introduction of the treatment might not reduce sufficiently the risk associated with the mitigated threat scenario, and thus the risk analyst decides to not update the risk model with the new treatment; or it can lead to the removal of another treatment because the new treatment reduces the level of risk of a threat scenario which was mitigated by the treatment. In this case, the requirement analyst needs to be informed about the removal of a treatment because it has to remove from the requirement model the task that is mapped to the treatment.

Example 4. The ADS-B introduction increases the availability of surveillance data. Thus, the likelihood of the threat scenario Failure of A/C tracking in the CORAS model is reduced. Consequently, the risk analyst determines that the treatment Run fault tolerant MRT is no longer needed and therefore removes it from the CORAS model. This treatment is mapped to the corresponding task in the requirement model since it protects the soft goal Availability of radar data, and hence the more general soft goal Availability of surveillance data. Therefore, the removal of treatment Run fault tolerant MRT in the CORAS model leads to the removal of the mapped task Run fault tolerant MRT. In Figure 4 the removal of elements is indicated by the diagonally striped elements.

TreatmentToTask. The interaction between the risk analyst and the requirement analyst is also required when the likelihood or the consequence scale changes or new threats emerge. For example, when a new security incident is reported, the risk analyst has to consider new threat scenarios for the damaged asset, and evaluate the risk associated with them. If the level of risk associated with the new threat scenarios is high,

the risk analyst identifies treatments that aim to reduce the risk to an acceptable level. The requirement analyst has to be notified about the new treatments, so that he/she can decide whether to implement the treatments or not. If the treatments provide sufficient risk reduction and are cost effective, the requirement analyst updates the requirement model with new tasks linked to the soft goal protecting the asset harmed by the threat mitigated by the treatments.

Example 5. Due to space constraints we do not show the modeling of this example, which in any case should be straightforward. A risk assessment is conducted that identifies the new threat scenario Eavesdropping on ADS-B communication. The risk of leakage of critical A/C position data due to ADS-B eavesdropping is considered as very unlikely since this kind of data is not sensitive, but the consequence of leakage of critical data is considered as a major consequence. The resulting risk level is therefore unacceptable. The treatment Implement encryption of ADS-B signal is identified for the threat scenario Eavesdropping on ADS-B communication. The addition of the treatment Implement encryption of ADS-B signal leads to the addition of a) a new soft goal Confidentiality of ADS-B signal which protects the resource ADS-B signal, b) a task Encryption of ADS-B signal which contributes to the fulfillment of the soft goal Confidentiality of ADS-B signal.

With these examples we see how the change-driven interplay works in practice. It supports the underlying idea of separation of concern between the two domains where the respective processes are conducted separately, yet sufficiently orchestrated to manage the changes in a coherent way. Moreover, the interplay ensures that consistency is maintained between the two domains and that none of the models become obsolete with respect to the other under change.

5 Related Work

The approach presented in this paper is related to works that have investigated the problems of orchestrating the requirement elicitation and analysis process with risk assessment process, and to works on how to handle change propagation.

Requirements and Risk Analysis Interplay. Several requirement frameworks have attempted to extend requirements conceptual models with security related concepts and to include risk analysis as part of the requirement analysis. Among goal-oriented approaches, van Lamsweerde extends KAOS by introducing the notions of obstacle [28] and anti-goal [15] to analyze the security concerns of a system. KAOS obstacle captures an undesired state of affairs that might harm safety goals (i.e., hazard) or threaten security goals (i.e., threat). KAOS anti-goal captures the intention of an attacker or considers it as a malicious obstacle. A formal framework is proposed to identify the obstacles to a goal in a given domain properties and to generate countermeasures to those obstacles. Liu et al. [18] proposes an extension of the i* framework [30] to identify attackers, and analyze vulnerabilities through actor's dependency links. In this framework, all actors are considered as potential attackers, and therefore their capabilities are analyzed and possible damages caused by actors are assessed. In Li et al. [16], the authors propose a formal framework to support the attacker analysis. Similarly, Elahi et al. [8] propose

extensions to i* to model and analyze the vulnerabilities affecting system requirements. Mayer et al. [24] propose a conceptual model for managing security of an information system based on several security methods (e.g., CORAS, ISO 27001). Asnar et al. [2] propose a concrete methodology, namely the Goal-Risk framework to analyze and model security problems. GR frameworks captures the stakeholders' goals, risks that might threaten the goals, and countermeasures required to mitigate the unacceptable the risk.

Compared to these approaches, the work presented in this paper proposes an interplay between requirement engineering process and risk assessment process that is based on *orchestration* rather than on *integration* of the two processes. Orchestration has has several advantages with respect to integration. The first one is that the requirement analyst and the risk analyst do not need to have in-depth expertise in the respective domains: they just need to know the mapped concepts on which the orchestration is based. Another key aspect of our approach is that the requirement and risk model are synchronized not on the basis of a periodic review but as soon a change is applied to the models. Thus, the orchestrated process ensures bidirectional consistency of requirement and the risk models.

Change propagation. Chechik et al. [4] propose a model-based approach to propagate changes between requirements and design models that utilize the relationship between the models to automatically propagate changes. Lin et al. [17] propose capturing requirement changes as a series of atomic changes in specifications and using algorithms to relate changes in requirements to corresponding changes in specifications.

With respect to change management for risk, the ISO 31000 standard [13] prescribes that change detection and identification for emerging risks should be conducted as part of the overall risk management process, but gives no specific guidelines on how to do this in practice. The well-known OCTAVE [1] risk assessment methodology recommends reviewing risks and critical assets, but offers no techniques or modeling for supporting the update of the risk assessment results. The approaches of Sherer [26] and Lund et al. [20] provide some support for maintenance of risk assessment results in the sense of restoring validity of risk documentation after changes, but change propagation and change impact analysis are not explicitly supported.

Other works relevant to change propagation are the one about the *generation and maintenance of traceability links*, and *model-to-model transformations*. Most of the works on the maintenance of traceability matrix focus on the recovery of traceability links between requirements and artifacts of different types e.g. code [6, 7, 14, 19], as in many cases these links are not explicitly represented; and on methods and CASE tools for the representation and management [5, 11, 12, 14] of traceability links.

Model-to-model transformation techniques such as VIATRA2 [29], QVT [27], and ATLAS [3] support change propagation by means of bidirectional incremental model synchronization.

In this paper we rely on VIATRA2 transformation framework to represent as graph transformation rules the mappings between concepts of SI* and CORAS. VIATRA2 ensures the automatic creation of traceability links between CORAS and SI* models and the execution of the mappings when a change affects a mapped concept.

6 Conclusions

In this paper we have presented an approach for handling change propagation and analyzing the mutual impact between the requirement engineering and risk analysis domains. Change propagation is based on conceptual mappings between the two domains and a set of well-defined mapping rules that handle the change propagation at the model level.

An important advantage of the approach is *separation of concerns*. Enforcing this principle has the advantage that the respective processes can leverage on each other while being conducted separately, and that the risk analysts do not need thorough expertise in the requirement domain or the requirement modeling language, and vice versa. They only need to have a high-level understanding of the mapped concepts. The change-driven interplay is supported by the formalization of the mappings as VIATRA2 graph transformation rules. The execution of the rules ensures the preservation of model consistency between the two domains under change.

In this paper the approach has been applied to the SI* requirement framework and the CORAS risk framework, although it is applicable also to alternative (competing) instantiations. Indeed, the concepts that are mapped in the former – such as goal, resource, and task – are in common to other goal-oriented approaches to requirement engineering. The same holds for asset and treatments that are key concepts in other risk analysis approaches. Thus, the validity of our approach goes beyond the demonstrated instantiations in this paper; alternative approaches to requirement engineering and risk analysis with similar underlying conceptual frameworks can be orchestrated by following the same approach.

In risk analysis, one of the objectives is to identify and document treatments the implementation of which ensures that the residual risk level is within the acceptable threshold, whereas requirements engineering aims at achieving the identified goals to an acceptable level. Thus, as future work, we plan to extend the framework with mapping rules for relating the residual risk level with the achievement level such that propagations are triggered also by changes to these. We moreover plan to extend the framework in order to handle more complex changes than the atomic ones. A further topic for future work is the definition of methods to handle conflicting changes such as a new requirement contradicting existing treatments.

Acknowledgments. This work has been partially supported by the European Commission under the 7th Framework Programme via the SecureChange (231101) project and the NESSoS (256980) network of excellence.

References

1. Alberts, C.J., Davey, J.: OCTAVE criteria version 2.0. Technical report CMU/SEI-2001-TR-016, Carnegie Mellon University (2004)
2. Asnar, Y., Giorgini, P., Mylopoulos, J.: Goal-driven risk assessment in requirements engineering. Requirements Engineering 16(2), 101–116 (2011)

3. Bzivin, J., Dup, G., Jouault, F., Pitette, G., Rougui, J.E.: First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In: 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture (2003)
4. Chechik, M., Lai, W., Nejati, S., Cabot, J., Diskin, Z., Easterbrook, S., Sabetzadeh, M., Salay, R.: Relationship-based change propagation: A case study. In: Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering. pp. 7–12. MISE'09, IEEE Computer Society, Washington, DC, USA (2009)
5. Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th International Conference on Software Engineering. pp. 362–371. ICSE '05, ACM, New York, NY, USA (2005)
6. Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: A feasibility study of automated natural language requirements analysis in market-driven development. Requir. Eng. 7(1), 20–33 (2002)
7. Egyed, A., Grünbacher, P.: Automating requirements traceability: Beyond the record & replay paradigm. In: ASE. pp. 163–171 (2002)
8. Elahi, G., Yu, E., Zannone, N.: A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Requirements Engineering 15(1), 41–62 (2009)
9. EUROCONTROL: ATM Strategy for the Years 2000+ (2003)
10. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering for trust management: model, methodology, and reasoning. International Journal of Information Security 5(4), 257–274 (2006)
11. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans. Softw. Eng. 32, 4–19 (January 2006)
12. IBM Rational DOORS, http://www-01.ibm.com/software/awdtools/doors/features/
13. ISO: ISO 31000 Risk management – Principles and guidelines (2009)
14. von Knethen, A., Grund, M.: Quatrace: A tool environment for (semi-) automatic impact analysis based on traces. In: ICSM. pp. 246–255 (2003)
15. Lamsweerde, A.V.: Elaborating security requirements by construction of intentional anti-models. In: Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on. pp. 148–157 (2004)
16. Li, T., Liu, L., Bryant, B.R.: Service Security Analysis Based on i*: An Approach from the Attacker Viewpoint. In: Security, Trust, and Privacy for Software Applications (STPSA 2010). pp. 127–133. Seoul (2010)
17. Lin, L., Prowell, S.J., Poore, J.H.: The impact of requirements changes on specifications and state machines. Softw. Pract. Exper. 39, 573–610 (2009)
18. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. Proc.of RE 3, 151–161 (2003)
19. Lucia, A.D., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. 16 (2007)
20. Lund, M.S., den Braber, F., Stølen, K.: Maintaining results from security assessments. In: 7th European Conference on Software Maintenance and Reengineering. pp. 341–350. IEEE Computer Society (2003)
21. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis - The CORAS Approach. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
22. Massacci, F., Mylopoulos, J., Zannone, N.: Security Requirements Engineering : The SI* Modeling Language and the Secure Tropos Methodology. In: Ras, Z., Tsay, L.S. (eds.) Advances in Intelligent Information Systems, Studies in Computational Intelligence, vol. 265, pp. 147–174. Springer Berlin / Heidelberg (2010)

23. Matulevicius, R., Mayer, N., Mouratidis, H., Dubois, E., Heymans, P., Genon, N.: Adapting Secure Tropos for security risk management in the early phases of information systems development. In: Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 5074, pp. 541–555. Springer Berlin / Heidelberg (2008)
24. Mayer, N., Heymans, P., Matulevicius, R.: Design of a modelling language for information system security risk management. In: Proceedings of the 1st International Conference on Research Challenges in Information Science (RCIS 2007). pp. 121–131 (2007)
25. Røstad, L.: An extended misuse case notation: Including vulnerabilities and the insider threat. In: The Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06) (2006)
26. Sherer, S.A.: Using risk analysis to manage software maintenance. J. Softw. Maint.: Res. Pract. 9(6), 345–364 (1997)
27. Stevens, P.: Bidirectional model transformations in QVT: Semantic issues and open questions. In: MoDELS. pp. 1–15 (2007)
28. Van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Transactions on Software Engineering 26(10), 978–1005 (2000)
29. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Science of Computer Programming 68(3), 214–234 (2007)
30. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto, Canada (1995)

APPENDIX C

Risk and Argument: A Risk-based Argumentation Method for Practical Security

Virginia N. L. Franqueira*, Thein Than Tun†, Yijun Yu†, Roel Wieringa* and Bashar Nuseibeh†‡
* University of Twente, Enschede, The Netherlands
Email: {franqueirav, r.j.wieringa}@ewi.utwente.nl
† The Open University, Milton Keynes, UK
Email: {t.t.tun, y.yu, b.nuseibeh}@open.ac.uk
‡ Lero, Irish Software Engineering Research Centre, Limerick, Ireland
Email: bashar.nuseibeh@lero.ie

***Abstract*—When showing that a software system meets certain security requirements, it is often necessary to work with formal and informal descriptions of the system behavior, vulnerabilities, and threats from potential attackers. In earlier work, Haley et al. [1] showed structured argumentation could deal with such mixed descriptions. However, incomplete and uncertain information, and limited resources force practitioners to settle for good-enough security. To deal with these conditions of practice, we extend the method of Haley et al. with risk assessment. The proposed method, RISA (RIsk assessment in Security Argumentation), uses public catalogs of security expertise to support the risk assessment, and to guide the security argumentation in identifying rebuttals and mitigations for security requirements satisfaction. We illustrate RISA with a realistic example of PIN Entry Device.**

***Index Terms*—Security Requirements, Argumentation, Risk Assessment, Common Attack Pattern Enumeration and Classification (CAPEC), Common Weakness Enumeration (CWE)**

I. Introduction

Structures of argumentation, such as those proposed by Toulmin et al. [2], provide an effective way to organize knowledge when convincing an audience about a claim. They have been used in many ways including to build safety cases [3], to demonstrate compliance to laws and regulations [4], and to show security requirements satisfaction [1]. Depending on the nature of the claim, arguments can be built upon observable and measurable evidence, such as the frequency of system failures, defects detected by code analysis tools and developers' certification records. Engineering of secure systems has many practical limitations, including incomplete knowledge about potential attackers, uncertainties about the system context, limited resources and different trade-off agendas. All in all, security is a non-zero-sum game between defenders and attackers of a system. Absolute security is usually not feasible, if at all possible. Argumentation works well in conditions of complete information and sufficient resources, but when one of these is lacking, it needs to be supplemented with risk assessment techniques that allow for uncertainty and incompleteness of available evidence, and allow partial application when resources are limited. It facilitates the achievement of *practical security*: "good-enough satisfaction" of security requirements within an "as low as reasonably practicable" [5] level of risk.

Haley et al. [1] have already suggested, but not elaborated, the need to relate argumentation and risk assessment for security. Extending that work, we present the RISA (RIsk-based Security Argumentation) method, which takes advantage of public catalogs of security expertise and empirical evidence, in particular, the CAPEC (Common Attack Pattern Enumeration and Classification), and the CWE (Common Weakness Enumeration) catalogs. The aims of RISA are to:

1) identify the risks to the satisfaction of security requirements;
2) differentiate between the risks that should be mitigated by the system context and the risks that should be mitigated by the system;
3) analyze those risks to be mitigated by the system and prioritize the risks found through arguments.

The main contribution of this paper is a systematic approach to assessing the risks associated with security arguments. This approach enables requirements engineers to make informed decisions about the implications of security risks, and how the system could maintain a good-enough level of security satisfaction. Our approach is different from other risk assessment frameworks in several ways. First, our approach uses security arguments, derived from a systematic description of the system context and formal reasoning about the satisfaction of the security requirements, as top-level structure of risk assessment. Second, RISA treats assumptions made in the arguments as a major source of risks, mitigation of which is a responsibility to be discharged either by the software or the system context. Third, the approach makes extensive use of publicly available catalogs on security risks, and mechanisms to mitigate risks. These evolving catalogs reflect the changing nature of security threats, and provide a valuable source of knowledge to requirements engineers.

The rest of the paper is organized as follows. Section II introduces our running example, the PIN Entry Device (PED) case study. Section III provides some background discussion on security requirements satisfaction, argumentation, and the security catalogs CAPEC and CWE that support the RISA method. Section IV presents the RISA method, describes the role played by argumentation and risk assessment, and lists its

steps. Section V demonstrates the RISA steps using the PED example. Section VI discusses the method and reviews related work, and Section VII concludes the paper.

II. Illustrative Case Study: PIN Entry Device

PIN Entry Device (PED) is a type of device widely-deployed and used by consumers to pay for goods with debit or credit smartcards at the Points-Of-Sales (POS).

When using the device, cardholders typically insert their cards, issued by a financial institution, into a card-reader interface of the PED, enter the PIN using the PED's keypad, and confirm the transaction value via a display on the PED itself. Then smartcard-based systems are expected to authenticate cardholders via the PIN and verify the card details against a public-key certificate before transactions can be completed successfully. These certificates are usually stored on the chip, but they can also be stored on the magnetic strip for compatibility with card-readers that have not adopted this technology. Most PEDs used in Europe implement the EMV (EuroPay, MasterCard and Visa) protocol in the process of authentication and authorization of payment transactions. This protocol drives the communication at the PED-card interface and the PED-bank interface. The protocol in principle allows only encrypted transmission of the PIN across these interfaces when the PED, card and bank support asymmetric cryptography. However, many card issuers adopt a low-cost EMV option in their smartcards that can be triggered to transmit unencrypted PIN on the interface PED-card.

Drimer et al. [6] studied two popular PEDs produced by different manufacturers, and evaluated either under the Visa evaluation scheme [7] or under the Common Criteria evaluation scheme [8]. The PEDs were found to be vulnerable to unsophisticated attacks in practice, and the authors have drawn lessons from this not only in relation to the evaluation process [6], [9] but also in relation to the software development life cycle. Throughout this paper, we will use this non-trivial example to explain and to motivate the RISA method.

III. Background

The RISA method relies on two main concepts proposed by Haley et al. [1]: the notion of the *satisfaction of security requirements*, and the use of *outer* and *inner* arguments to demonstrate the system security. This section recaps those concepts, and discusses the security catalogs used by the RISA method.

A. Satisfaction of Security Requirements

Following the WSR principle of the Problem Frames approach [10], the framework of Haley et al. separates software artifacts into W, S and R, where W represents a description of the world in which the software is to be used (i.e., the system context), S represents the specification of a system, and R represents a description of the requirements. The main property of these artifacts is that the software within the system context should satisfy the requirements, as indicated by the entailment (1):

$$W, S \vdash R \tag{1}$$

Similar to the Problem Frames approach, the framework of Haley et al. describes the world context W in terms of *domains* (short for *problem world domains*), elements of the world that can be either tangible (such as people, other systems, and hardware) or intangible (such as software and data structure). Typically, W also contains assumptions made about these domains. Domains interface with each other and with the system S via *shared phenomena* (such as events, states and values) that are controlled by a domain and are visible to some other domain(s).

In the framework of Haley et al., security requirements are constraints on functional requirements that protect the assets from identified harms. For example, in the requirement "Provide Human Resource (HR) data requested by the user, only to HR staff", providing HR data to users making the requests is regarded as a functional requirement, and ensuring that only those requests from members of HR staff are fulfilled is regarded as a constraint on the functional requirement, thus a security requirement. Security requirements are part of R in the entailment (1). Therefore, satisfaction of security requirements, spelled out in *security arguments*, show (i) whether properties of W and S entail the security requirements, and (ii) whether assumptions in W and S are correct.

B. Security Arguments of Haley et al. Framework

The framework of Haley et al. distinguishes two kinds of argument for satisfaction of security requirements.

1) Outer arguments: The *outer arguments* show whether properties of W and S entail the security requirements. These arguments are typically expressed in a formal language, such as propositional logic. Therefore, outer arguments are proofs of the entailment (1) for security requirements. This is expressed as follows:

$$(domain\ behavior\ premises) \vdash (security\ requirement(s)) \tag{2}$$

Outer arguments rely on properties of W and S (*domain behavior premises*), some of which may turn out to be incorrect assumptions. These premises need to be challenged, and be grounded in facts if possible, or taken as true, for instance, on the basis of a lack of contrary evidence.

2) Inner arguments: The general purpose of an *inner argument* is to try to rebut an outer argument by questioning its premises. Notice that the outer arguments establish the scope of security assessment, whilst the inner arguments deepen the assessment. The framework of Haley et al. uses Toulmin-style structures, enhanced with the notion of recursiveness from Newman et al. [11], for inner arguments. The general structure of inner arguments used in the framework of Haley et al. is shown in Fig. 1.

A **claim** is the object of an argument, a statement that one wishes to convince an audience to accept. A **ground** is a piece of evidence, a fact, a theory, a phenomenon considered to be true. A **warrant** is a statement that links a ground and a claim,

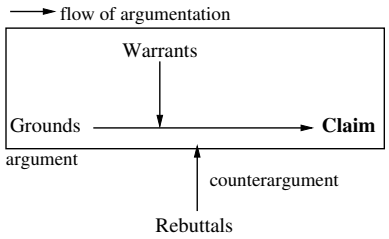


Fig. 1. Structure of arguments used in the framework of Haley et al. [1]

showing how a claim is justified by ground(s). A **rebuttal** is a counterargument which reduces support for the claim. Specifically in the case of security-related argumentation, rebuttals represent the *risks* that the claim of an argument is false. Rebuttals can be mitigated in order to restore the confidence that the claim of the rebutted argument is true. A mitigation, while negating a rebuttal, may introduce additional knowledge to show that the rebuttal, i.e. a risk, can somehow be tolerated. Therefore, mitigations address risks but may not necessarily eliminate the risks, and residual risks may remain. Moreover, mitigations may also introduce new risks, leading to new rounds of rebuttals and mitigations in argumentation.

C. Relationship between Outer and Inner Arguments

Fig. 2 shows how outer and inner arguments are related: the formal outer argument provides the main structure that drives the inner argumentation. Each of the outer argument premises $(a \rightarrow b)$ is the beginning for one thread of inner arguments, where a is the ground and b is the claim to be challenged by examining the ground a and the implication $\rightarrow$. Each round of inner argumentation is indicated by the notation $R \mid Mr.n$, where R stands for Risk and M for Mitigation, r indicates the round and n represents a sequential number.

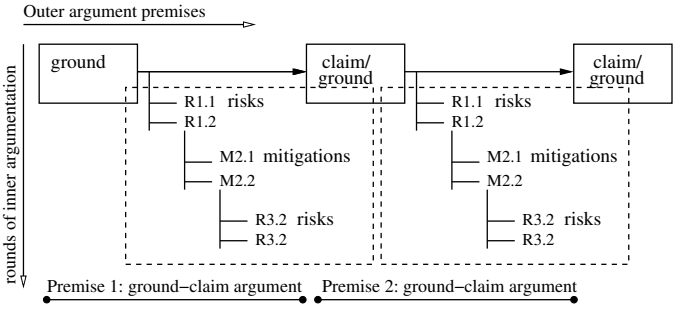


Fig. 2. Relationship between outer and inner arguments

In Section IV, we contrast the process of argumentation used in the framework of Haley et al. with our new approach proposed by the RISA method.

D. Public Catalogs of Security Expertise

The National Cyber Security Division of the U.S. Department of Homeland Security has a strategic initiative to promote software assurance. Two projects under this initiative are particularly useful for the RISA method, namely, CAPEC and CWE[1].

[1]Their websites, http://cwe.mitre.org/ and http://capec.mitre.org/, are both sponsored and managed by the Mitre Corporation.

Common Attack Pattern Enumeration and Classification (CAPEC) is a publicly available catalog of known attacks, described according to a standard schema [12]. The CAPEC catalog[2], contains 460 entries, organized according to different views and categories. Complete CAPEC entries provide not only information about attack execution flow, methods of attack, attack prerequisites, and attacker skills/knowledge/resources required, but also about typical severity, and generic solutions and mitigations. In addition, complete entries in CAPEC also provide pointers to specific weaknesses ("underlying issues that may cause vulnerabilities" [12]), i.e. to CWE(s), and to concrete examples of vulnerabilities which have been detected in use (these are CVE (Common Vulnerabilities and Exposure) entries recorded in the National Vulnerability Database[3]).

Common Weakness Enumeration (CWE) is a public catalog of weaknesses[4], where each weakness can either become a direct source of vulnerabilities, or contribute indirectly to an increase in the likelihood of an attack to happen and/or an increase in the impact of the attack, if it succeeds [12]. It also follows a standard schema [13], providing different views, groupings and relations with other weaknesses, as well as pointers to CAPEC and CVE entries. Complete CWEs indicate common consequences, likelihood of exploit, detection methods and potential mitigations at different phases, such as Requirements and Architecture/Design. The RISA method uses CAPEC and CWE as sources for security knowledge, making the analysis of risks more systematic, repeatable, and less dependent on subjective judgments.

IV. Overview of the RISA Method

As shown in Fig. 3 the RISA (RIsk assessment in Security Argumentation) method extends the process of argumentation for security requirements proposed in the Haley et al. framework by incorporating a process of risk assessment.

Steps 1 to 3 of the proposed approach are same as the first three steps of the framework of Haley et al., and are briefly summarized below.

A. Step 1 to Step 3

In Step 1 (Identify Functional Requirements), functional requirements of the system and the system context (domains and shared phenomena) are identified. These requirements may be derived from the higher-level goals of the system. In Step 2 (Identify Security Goals), assets that need to be protected, management principles regarding those assets, and security goals are identified. In Step 3 (Identify Security Requirements), security requirements are derived from security goals, and are expressed as constraints on the functional requirements identified in Step 1. Problem diagrams are constructed in this step.

[2]http://capec.mitre.org/data/, version 1.6, accessed on 21 Feb 2011
[3]http://nvd.nist.gov/, version 2.0, accessed on 21 Feb 2011
[4]http://cwe.mitre.org/data/, version 1.11, accessed on 21 Feb 2011

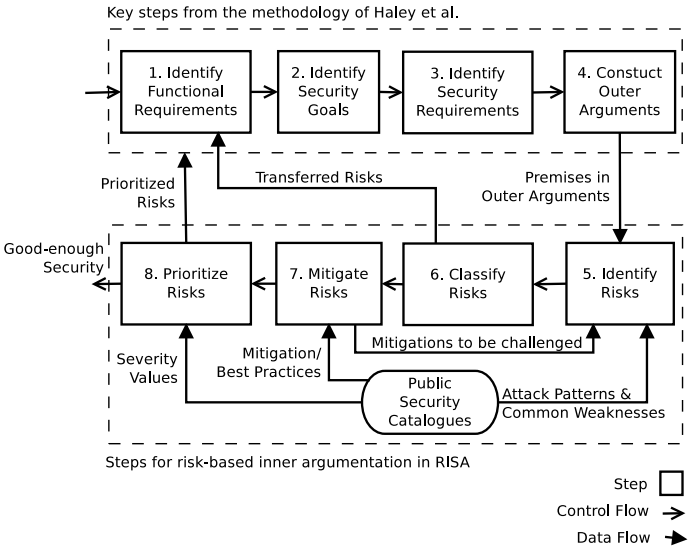


Fig. 3. Schematic overview of the RISA method

B. Step 4: Construct Outer Arguments

Unlike the fourth step of the Haley et al. framework, only the outer arguments for security requirements (excluding the inner arguments) are constructed in Step 4 of RISA. These outer arguments are formal, and they make use of domain properties, correctness of which is examined by inner arguments. Behavioral premises used in the outer arguments may represent risks, which are identified using a systematic risk assessment process in RISA. This is represented in the figure by the arrow from Step 4 to Step 5.

Steps 5 to 8 correspond with the process of constructing inner arguments in the Haley et al. framework. These four steps show how domain assumptions in outer arguments are challenged by means of risk assessment based on public security catalogs.

C. Step 5: Identify Risks

In this step, behavioral premises in outer arguments regarding the domains (arrow from Step 4 to Step 5 in Fig. 3) are identified as potential risks. For instance, in the PED example, there could be a behavioral premise about the confidentiality of the PIN entered using the keypad. Public security catalogs are then searched to find known security weaknesses regarding the confidentiality of passwords entered using a keypad.

D. Step 6: Classify Risks

The risks are classified into two groups. In one group are risks transferred to the context because nothing can be done by the system to mitigate them. In the PED example, the confidentiality of the PIN (one of the PED security requirements) depends on cards having certain cryptographic capabilities. Although the PED can generally comply with this demand, there is still the risk that the card will not comply. Therefore, the obligation to mitigate this risk is transferred either partially or fully to the card. In the other group are risks that should be mitigated by the system. In the PED example, integrity of the PIN over the network has to be ensured by the system. Classification and transferring of obligations to mitigate risks to the system and its context could modify the behavior and properties of the domains and shared phenomena, and perhaps even the functional requirements (as indicated by the arrow from Step 6 to Step 1). As a result of these changes, outer arguments may be rebutted.

E. Step 7: Mitigate Risks

Appropriate security mechanisms for mitigating the risks are searched for in the public security catalogs (arrow from the catalogs to Step 7). Some of these mechanisms themselves could introduce new risks and therefore should be assessed in a new round of inner argumentation (arrow from Step 7 to Step 5).

F. Step 8: Prioritize Risks

In the last step, risks are prioritized on the basis of their severity as indicated by the public security catalogs (arrow from catalogs to Step 8). These risks affect the priority of requirements to be satisfied (arrow from Step 8 to Steps 1–4). When the residual risks are deemed to be acceptable given the limitation of development resources, the system has reached the level of good-enough security.

G. Discussion

Fig. 4 illustrates how elements from risk assessment, used in the RISA method, fit into the inner argument schema, as presented in Fig. 2. Unlike the traditional process of argumentation (based on Toulmin's practical argumentation [2]), that is intrinsically performed in depth-first as a dialog (Fig. 2), argumentation based on risk assessment is performed in breadth-first. Each of these approaches have advantages (discussed in Section VI) but, typically, in risk assessment two rounds of argumentation are performed within the same cycle: one is related to risks that rebut outer arguments, and the other is related to mitigations that counter these risks.

Therefore, risks regarding all behavioral premises are identified and classified in terms of those that should be mitigated by the system and those that should be mitigated by the context, and prioritized on the basis of the severity of the risk they represent, providing input to the next round of argumentation.

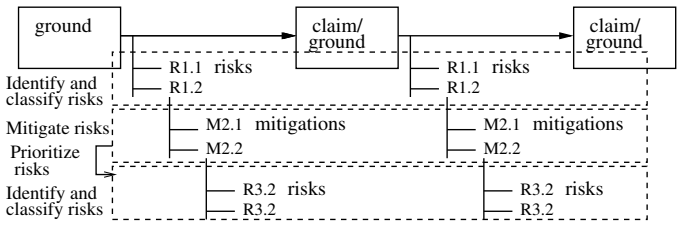


Fig. 4. Risk-based inner argumentation

The recursiveness of the inner argumentation is represented by the indirect connection between Step 5 and Step 7, which involves the process of finding mitigations to risks, and the

direct connection between Step 7 and Step 5, which involves the process of finding new risks in mitigations.

Public catalogs provide input for all steps in risk assessment, except for Step 6, which has to be done by domain experts relying on the knowledge of the exact requirements. In the RISA method, we use CAPEC and CWE to feed the identification of risks with descriptions and information about known attack patterns and weaknesses in software. They can also provide information on how these attacks and weaknesses can be mitigated, and empirical values indicating the severity that allow the prioritization of risks. In some cases, in-house security catalogs may supplement public security catalogs.

Since these risks are attached to arguments and security requirements, prioritizing risks indirectly results in the prioritization of arguments and security requirements.

V. The PED Example

This section describes a step-by-step application of RISA to the PED example.

A. Satisfaction of Security Requirements

The system under analysis is the PED; it consists of four main components: card-reader, keypad, display and CPU.

1) Step 1—Identify Functional Requirements: The overall functional goal of the system in relation to PED users is the following:

[FG1]: Provide convenient payment option at Points-Of-Sale to consumers

The following functional requirement can be derived from the functional goal above:

[FR1]: Allow consumers to pay at Points-Of-Sale with PIN

2) Step 2—Identify Security Goals: Obviously, the consumers' PIN is the most valuable asset in this case. Protecting the PIN is important not only because of the potential financial impact on the consumers, but also because of the substantial negative impact it will have on the reputation of the banking authorities [14], and the PED manufacturers. Card details, stored in smartcards, are also relevant assets; their value comes from the possibility to fake the magnetic-strip on the smartcards [6]. Other assets involved with the PED system include: transaction value, design characteristics, the smartcard itself, and cryptographic keys. For the illustration of our analysis, the main security goal is to protect the PIN.

3) Step 3—Identify Security Requirements: As mentioned earlier, security requirements are viewed as constraints on the system functional requirement according to its functional goals. In this case, such a composition produces the following two security requirements:

[SR1]: PIN entered by consumers shall remain confidential during payment transactions at Points-Of-Sale [SR2]: PIN entered by consumers shall remain accurate during payment transactions at Points-Of-Sale (i.e. integrity of the PIN shall be preserved)

According to the documentation available about the PED system (e.g. [6], [14]), these two requirements are satisfied by implementing the following security functions:

[SF1]: Enclosure of PED components provides tamper detection and response mechanisms to resist physical attacks [SF2]: Encryption/Decryption of PIN ensures that the PIN is encrypted within the PED immediately after the PIN entry is complete

These security functions correspond with the "requirements" A1.1 and C2, extracted from the PED security requirements document from Mastercard [15].

The system context, W in the entailment (1), is then elaborated. The functional requirement of the PED helps us to delimit the context; from "payment transaction using a PED" we identify five domains: consumer, card, merchant, terminal and bank. Using a slightly modified version of the notation used by the Problem Frames approach [10], Fig. 5 shows the context of the PED system and its security requirements. The notation is unusual in two ways: (i) it treats the PED system as a machine with its own components, and (ii) causality in shared phenomena is indicated by directed arrows. Notice that the diagram shows the shared phenomena related not only to PIN, but also to the card details and the transaction value, which are relevant to the PED payment transactions.

In terms of PIN, the diagram illustrates the behavior already described in Section II, namely that "Cardholders [consumers] typically insert their cards ... into the PED's card-reader interface, enter their PIN using the PED's keypad, and confirm the transaction value via a display on the PED itself". Note that the action performed by consumers to insert their cards into the card-reader is not represented explicitly in the diagram.

The PED system has two possible main usage scenarios. It can be used at the Points-Of-Sales connected to a terminal, or stand-alone. The former case is often found in supermarkets, once the merchant scans products, their value get registered by the terminal; at the end, the terminal sends the value of the transaction to the PED that displays the value to the consumer. The latter case is often found in restaurants: the merchant enters the transaction value directly into the PED via the keypad and the PED displays this value to the consumer.

The card details flow from their initial location (the card) along with the PIN and the transaction value until they reach the bank for approval.

The shared phenomena related to the PIN in Fig. 5 are detailed using the following convention: an arrow from a domain A annotated with phenomenon b is written as $A!b$.

- consumer!PIN: consumer enters PIN
- keypad!PIN: keypad sends PIN to the card-reader
- card-reader!PIN: card-readers sends PIN to the card
- card!confirmation-PIN-ok: card requests confirmation that PIN is ok to the card-reader
- card-reader!PIN-confirmed(PIN,card-details): card-reader sends confirmation that PIN is ok to the PED CPU
- CPU!authorization-request(PIN,value,card-details): CPU sends request for authorization of payment transaction to bank
- bank!confirmation-transaction: bank sends confirmation (positive or negative) of transaction to the PED CPU

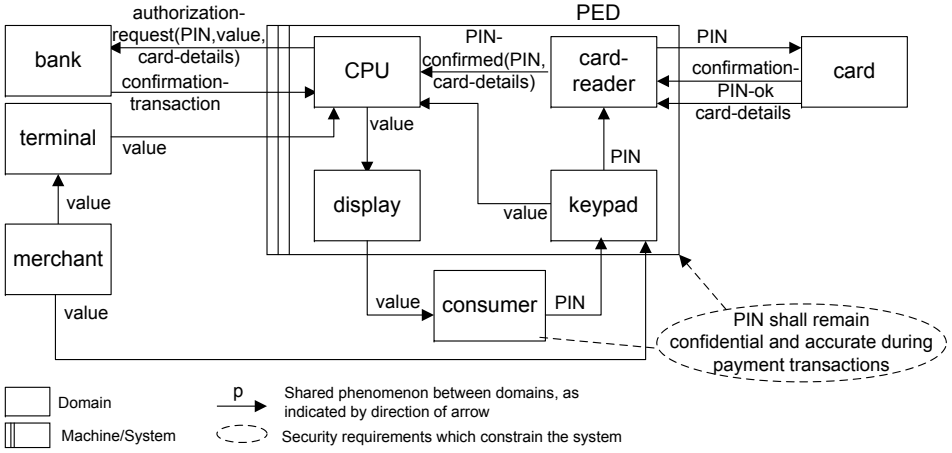


Fig. 5. System context of the PED system and its security requirements

4) Step 4—Construct Outer Arguments: The behavior of the PIN needs to guarantee that:

P1, P2, P3, P4, P5, P6, A7 ⊢ bank!confirmation-transaction

The premises are defined below using the propositional logic.

Premises:
P1. consumer!PIN → keypad!PIN
P2. keypad!PIN → card-reader!PIN
P3. card-reader!PIN → card!confirmation-PIN-ok
P4. card!confirmation-PIN-ok → card-reader!PIN-confirmed
P5. card-reader!PIN-confirmed → CPU!authorization-request
P6. CPU!authorization-request → bank!confirmation-transaction
Triggering assumption:
A7. consumer!PIN holds
Conclusions:
C8. keypad!PIN (Detach,P1,A7)
C9. card-reader!PIN (Detach,P2,C8)
C10. card!confirmation-PIN-ok (Detach,P3,C9)
C11. card-reader!PIN-confirmed (Detach,P4,C10)
C12. CPU!authorization-request (Detach,P5,C11)
C13. bank!confirmation-transaction (Detach,P6,C12)

Assuming that the phenomena in behavioral premises (P1–P6) are triggered in sequence, context is correct and later implementation of the PED does not introduce deviations from the behavior specified, this proof means that, *in principle*, the PED behavior (i.e., the PED with its security functions under its context) can satisfy both security requirements (SR1: confidentiality of PIN, and SR2: accuracy of PIN) while meeting its own functional requirement (FR1). It proves that the entailment (2) can be fulfilled for the PED example.

However, the proof premises (P1-P6) and triggering assumption (A7) can be challenged in practice. If any of these can be challenged effectively, the result could represent a non-satisfaction of entailment (2) for the PED example.

B. Risk assessment for inner arguments

The premises (P1–P6) and triggering assumption (A7) of the outer argument, the output of the previous step, provides a structure for assessing risks. The following discussion will focus on risks related to SR1 only.

1) Step 5—Identify Risks: This activity aims to identify the risks related to the security requirement SR1 that could rebut all the premises and triggering assumptions carried over from the outer argument. Supported by the CAPEC and CWE security catalogs, the activity involves searching for catalog entries that represent a risk to the claim of each premise, including the triggering assumption. For example, what has to be challenged for premise "P1: consumer!PIN → keypad!PIN" is the confidentiality of consumers' PIN (SR1). This involves: (i) the confidentiality of the PIN as it is entered by consumers (A7), and (ii) the confidentiality of the PIN from the moment it is entered by consumers until it reaches the keypad. Therefore, these two aspects drive the analysis of risks which can rebut P1. As illustrated in Fig. 6, risks identified for P1, given A7, allow us to evaluate the satisfaction of conclusion C8 of the outer argument for SR1. Similar rationale applies to P2-P6.

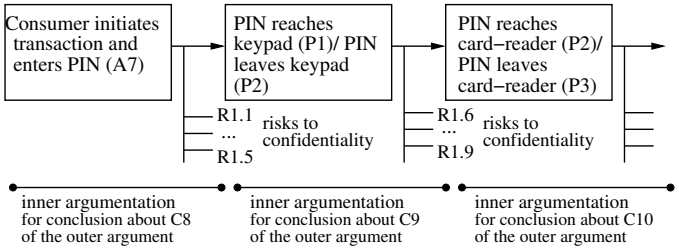


Fig. 6. Behavioral premises of the PED system to be challenged via risk-based inner argumentation

Table I lists the risks identified for premises P1 and P2 with references to CAPEC and CWE entries. It also contains some references marked with the symbol ⋆, indicating the CAPEC/CWE entries that are related to the risk but are not incomplete enough to be useful at this stage. Since these catalogs are constantly evolving it is important to keep them for future reference.

2) Step 6—Classify Risks: We classify risks according to two types of risk treatments.

TABLE I
IDENTIFICATION OF RISKS FOR PREMISES P1 AND P2

Challenged	Risk	Reference
Premise P1	R1.1: consumer is triggered to reveal PIN via social engineering attack	CAPEC-403⋆
Premise P1	R1.2: PIN is revealed by missing PIN field masking	CWE-549
Premise P1	R1.3: PIN is revealed by brute force attack	CAPEC-49, CAPEC-70 & CAPEC-112
Premise P1	R1.4: PIN is revealed due to lack of aging policy	CWE-262
Premise P1	R1.5: PIN is collected by fake PED set to allow pharming attack	CAPEC-89
Premise P2	R1.6: PIN is revealed if sent unencrypted within the PED and the PED enclosure can be tampered with	CWE-311 & CAPEC-436⋆
Premise P2	R1.7: PIN is revealed if sent encrypted within the PED but PED enclosure can be tampered with	CAPEC-20 & CWE-327 & CAPEC-436⋆
Premise P2	R1.8: PIN is revealed via sniffer installed by PED administrators	CAPEC-65
Premise P2	R1.9: Unauthorized access to PIN is concealed via log injection-tampering-forging by PED administrators	CAPEC-93

A risk is classified as "transfer risk" if it is assumed that context domains, involved in ensuring the satisfaction of the entailment (1), will be responsible for its mitigation. Identifying such a class of risks is important for two reasons: it allows different parties in the PED context domains to be made explicitly accountable for the mitigation of the identified risks, and it allows regulations to be enforced [16].

A risk is classified as "mitigate risk" if it is assumed that the system will be responsible for its mitigation. The classification of risks from Table I is shown in Table II. Note that this table illustrates with R1.7 that one risk can be classified in both classes. Typically, risks classified under "mitigate risk" contain brief descriptions of mitigations in the format of requirements. Mitigations described in this table are mostly retrieved from best practices *solutions and mitigations* described in the CAPEC and CWE entries mentioned in Table I. For example, CWE-311 (reference for risk R1.6), entitled "Missing Encryption of Sensitive Data" discusses potential mitigations for this weakness in terms of different phases of the software life cycle. Under *Phase:Requirements*, we find the description of mitigation "Any transmission of PIN should use well-vetted encryption algorithms", as presented in Table II.

3) Step 7—Mitigate Risks: Only those risks classified as "mitigate risks" are carried over to this stage. This step involves a cross-analysis of mitigations identified for each risk during the classification activity (Table II), in order to obtain

TABLE II
CLASSIFICATION OF RISKS FOR PREMISES P1 AND P2

Risk	Risk treatment
R1.1	**Transfer risk**: assumed consumers take mitigations
R1.2	**Mitigate risk**: PED should obfuscate display of PIN as entered by consumers in keypad
R1.3	**Transfer risk**: assumed banks (e.g., require strong PIN policy) and consumers (e.g., avoid common, guessable PIN) take mitigations
R1.4	**Transfer risk**: assumed banks and card issuers take mitigations (e.g., by periodically requiring PIN change and card renewal)
R1.5	**Mitigate risk**: PED should use (i) authentication mechanisms, and (ii) audit mechanisms to log authorized replacements
R1.6	**Mitigate risk**: Any transmission of PIN should use well-vetted encryption algorithms
R1.7	**Mitigate risk**: (i) encryption of PIN should use accepted algorithms and recommended key sizes, (ii) cryptographic keys should be managed and protected (**Transfer risk**: assumed cards will also comply with this mitigation), (iii) PED design should allow upgrade of cryptographic algorithms
R1.8	**Mitigate risk**: Any transmission of PIN should be encrypted
R1.9	**Mitigate risk**: PED should provide access control to physical log files

a consolidated list of mitigations, as shown in Table III. This table shows that mitigation M2.4 counters risks R1.6, R1.7 and R1.8, and that risk R1.7 is countered by mitigations M2.4, M2.5 and M2.6. Notice that risks R1.1., R1.3 and R1.4 are not presented in the table.

TABLE III
MITIGATIONS OF RISKS IDENTIFIED FOR P1 AND P2

Risk	Mitigation
R1.2	M2.1: PED should obfuscate display of PIN as entered by consumers in keypad
R1.5	M2.2: PED should use authentication mechanisms
R1.5	M2.3: PED should have audit mechanisms to log authorized replacements
R1.6 & R1.7 & R1.8	M2.4: Any transmission of PIN should use well-vetted encryption algorithms and recommended key sizes
R1.7	M2.5: Cryptographic keys should be managed and protected
R1.7	M2.6: PED design should allow upgrade of cryptographic algorithms
R1.9	M2.7: PED should provide access control to physical log files

4) Step 8—Prioritize Risks: This step involves retrieving from the CAPEC and CWE entries indicating the severity of risks identified, as shown in Table IV.

This table makes it evident that this analysis only gives a rough impression of the severity of risks and that security experts are again confronted with incomplete information, which often happens in security practice. It also becomes apparent that, because several CAPEC and CWE entries may

TABLE IV
PRIORITIZATION OF RISKS FOR PREMISES P1 AND P2

Mitigation	Risk	Typical risk severity
M2.1	R1.2	no indication in the CWE
M2.2	R1.5	very high
M2.3	R1.5	
M2.4	R1.6 & R1.7 & R1.8	low to very high (depending on specifics of different attacks)
M2.5	R1.7	low to high (depending on specifics of different attacks)
M2.6	R1.7	
M2.7	R1.9	high

refer to the same risk, we may obtain in the end a lower bound and an upper bound on risk severity. In this case, a decision should be made about the strategy to follow depending on several factors, such as attitude towards risk (e.g., risk-averse, risk-taking or in-between), security-criticality of the system, and so on. For example, from a risk-averse point-of-view, we consider M2.2, M2.3 and M2.4 as priority over the others.

Recursion of Inner Arguments Mitigations may also introduce new risks, so for each mitigation in Table III a new iteration of Step 5 may be required. These iterations stop when the system security is considered good-enough, i.e. the residual risks are considered acceptable, and the resources for security analysis have been used (e.g., deadline or budget have been reached).

For example, if we take mitigation M2.3 which is related to the very high severity risk R1.5 (Table IV), we can see that this risk refers to the possibility of pharming attacks (Table I). According to CAPEC-89, a pharming attack occurs "when the victim is fooled into entering sensitive data into supposedly trusted locations". This risk challenges the premise P1, which refers to the PIN shared between consumer and keypad (Fig. 5). Since SR1 is about the confidentiality of PIN, the risk assessment suggests that the pharming attack is a rebuttal to the security argument of SR1. The risk assessment further indicates that one way to mitigate this risk is by introducing an audit mechanism for the keypad (Table III). This new round of rebuttal to the premises of the security argument, and a mitigation to the rebuttal are obtained from the risk assessment.

VI. RELATED WORK & DISCUSSION

A. Risk in Requirements Engineering

In requirements engineering literature, the issue of risks has been considered in two related yet distinct ways: project risks and system risks. Project risks are related to the process of software development and factors that may contribute to the failure and success of the project. Several factors may be related to the project risks, including requirements creep [17], [18], requirements negotiation and project estimates [18], and project resources. System risks are related to the behavior of the software system that may contribute to the system satisfying or not satisfying the requirements. Factors contributing to system risks include missing or incorrect requirements and domain assumptions [19].

Threat modeling for the elicitation of security requirements has been extensively researched in the domain of requirements engineering. Published approaches include: misuse cases [20], abuse cases [21], attack trees [22], abuse frames [23], anti-goals [24], and combinations of these [25]. These approaches overlap only partially with the RISA method, mainly in the activity of risk identification. In this activity, they may provide a more in-depth analysis of specific issues/scenarios, therefore, in this sense they complement RISA. However, they lack the advantage of argumentation to allow validation of security requirements satisfaction and of maintaining the focus of security on the system as a whole.

B. Structured Argumentation

Argumentation provides a rationale to convince an audience that a claim should be considered valid. Three qualities are often discussed in the informal argumentation literature: convincingness, soundness, and completeness. Convincingness relates to whether the argumentation is compelling enough to assure an intended audience that the conclusion reached is reasonable [1]. Soundness relates to whether the argumentation fulfills the argumentation schema and is based on "true premises" [26]. Completeness relates to whether nothing has been omitted that could lead to a different conclusion about a claim [26], [27].

A known problem in argumentation is the subjectivity involved in identifying arguments and counterarguments (which relates to soundness), and the difficulty in determining completeness. Proposals to reduce these problems rely on the help of: (i) pre-defined critical questions [28], [29], (ii) what-if scenarios [30], (ii) expert assurance checks [26], (v) guidelines [31] or (vi) how/why questioning, as proposed in [1]. However, these approaches provide limited support and are rather static, i.e. they do not evolve in the speed required to assure good-enough security of systems. The RISA method reduces these problems by using ever evolving public catalogs, updated using input by a pool of security experts from several organizations[5].

Since security involves uncertainty and incomplete information, it becomes difficult, if not impossible, to show satisfaction of these qualities. Expert judgment is needed, probably enhanced with peer review [26], to improve the quality of *good-enough* security argumentation. Nevertheless, the main benefit of using argumentation is that, in contrast to an ad-hoc approach, it structures the reasoning and exposes it to criticism [26]. The RISA method contributes towards reducing incompleteness and uncertainty because its risk-based argumentation is supported by public catalogs which accumulate information about risks, best practice mitigations and empirical severity values based on input from a large community of security experts.

[5]http://cwe.mitre.org/community/index.html, accessed on 21 Feb 2011

C. Traditional versus risk-based (security) argumentation

As mentioned in Section III, the process of argumentation traditionally follows a depth-first style. It provides a neat and intuitive view about the evolution of an argument in the format of a debate between two opponents. It is well-suited to be represented as a tree structure of arguments and counterarguments.

This process of argumentation, however, is not suitable when reasoning about practical security. For example, it often happens that one mitigation counters several risks, one risk challenges several premises, and several mitigations introduce one same new risk as seen in Table III, Step 2. As a result, rather than a tree, a more complex graph structure of arguments is often needed. In such situations, a breadth-first style of argumentation based on risk assessment scales better. The RISA method is designed for such practical needs, while still providing the ability to reconstruct argument threads via backward traceability. Thus it is possible to link mitigations back to risks, then back to premises, which relate to the outer arguments to be validated in the first place.

D. Risk Assessment

Three basic elements distinguish the risk-based security argumentation method described in this paper from other risk assessment frameworks: (i) it provides a rationale for a systematic representation of system context from which the behavior of the system is derived and the outer argument is constructed, providing structure to the risk assessment part of the method, (ii) it makes assumptions about domains of the system context explicit via transferred risks, and (iii) it uses public security catalogs to support the risk assessment.

Explicit Context. In the CORAS framework [32], context is established by means of an asset diagram (CORAS Step 2) which contains all assets that are logically or physically related to the system to be assessed. Assets that are directly harmed as a consequence of an unwanted event affecting the target of evaluation are considered as part of the system under analysis: otherwise, they are considered as part of the system context. The boundary between system and system context can be further adjusted in CORAS (Step 3) after a preliminary analysis of risks by means of "dependent diagrams" [33]. Context is delimited in the RISA method by means of the system functional goal and security requirements, from which context domains are derived. This allows us to relate the results of risk assessment to the satisfaction of security requirements.

Other security risk assessment and evaluation frameworks (e.g., CRAMM [34], ISO 27005:2008 [35], AS/NZS4360:2004 [36], and Common Criteria [8]) do not prescribe any representation or delimitation rationale for context specification; context is often described in natural language.

Assumptions as Risk. Among the above mentioned risk assessment and evaluation frameworks, the Australian/New Zealand Standard (AS/NZS4360:2004 [36]) is the only one which conveys explicitly the idea that assumptions represent risks. It prescribes that assumptions should be recorded and clearly stated but, most importantly, mandates sensitivity analysis to test the effect of uncertainty in assumptions. The current RISA method does not incorporate validation of assumptions made about the system context but it classifies them as risks to be transferred. We plan to further study risks transferred and how they affect the satisfaction of security requirements as future work.

Evolving & Detailed Source of Information. The CORAS framework [32] uses the guidelines in ISO 27001:2008 [35] to support risk assessment. CRAMM [34] is supported by a proprietary database of security controls that are traceable to risks derived also from the ISO 27001:2008 [35]. Although ISO 27001 is publicly available, it is a static document and does not provide the level of details provided by CAPEC and CWE catalogs. However, both CAPEC and CWE, at their current stage, still need a lot of improvements, especially in their search capabilities. This is an area that has started to receive attention from researchers, see e.g. [37].

Specific RISA steps, such as "Identify Risks" and "Mitigate Risks", could be supported by other risk assessment frameworks such as the model-based approach of CORAS which uses icons instead of tables. However, although it may improve the user-friendliness of RISA, the traceability of several rounds of argumentation may become more complex.

VII. Conclusion and Future Work

Although absolute security is not possible in practice, security requirements still have to be satisfied to the extent allowed by incomplete information, uncertainty and limited resources. When dealing with practical security, requirements engineers need to reason about whether security is good enough. That reasoning typically involves risk assessment. Extending existing work on argumentation for security, the proposed RISA method has shown how argumentation can be extended with risk assessment, thereby exploiting the ability to identify, classify, mitigate and prioritize risks, and feeding the prioritized risks back to the process of reasoning about the satisfaction of the security requirements. RISA takes advantage of publicly available catalogs of common attacks and weaknesses, thus a degree of objectivity can be achieved in the risk assessment. Nevertheless, subjectivity is not completely eliminated by catalogs such as CWE and CAPEC: for instance, the evaluation of risk severity is likely to remain subjective. However, this evaluation is prone to scrutiny by security experts from the wider community.

The most pressing issue in our future work is validation. We have demonstrated an application of RISA to a realistic example of the PIN Entry Device. We are planning to apply the RISA method to significant industrial case studies, including the Air Traffic Management system [38]. Furthermore, we see two main directions for future work which complement each other: enhancements to the prioritization of risks and mitigations, consequently reflecting on the prioritization of arguments, and tool support.

In the current version of the RISA method, prioritization of risks is qualitative, coarse-grained, and is detached from

context. We believe that the Common Weakness Scoring System [39], which is another recent initiative related to the CWE, may be valuable in working towards a more systematic quantitative prioritization of risks taking into account factors related, for example, to security concerns particular to a business domain and technology. Prioritization could also be enhanced by incorporating cost/benefit trade-off considerations (as in [40]) about mitigations, as well as consideration about risks. Finally, uncertainty is another important aspect to be added into prioritization: early work on Probabilistic Argumentation [41] seems to be a promising starting point in this direction.

Decision making tool support would greatly improve the practical use of the RISA method. We plan to investigate how to provide effective automated reasoning about security requirements, based on feedback from risk-based argumentation.

ACKNOWLEDGMENT

The first author is supported by the research program Sentinels (http://www.sentinels.nl). The UK-based authors are supported by the SecureChange project, and SFI grant 03/CE2/I303_1. We thank the anonymous reviewers for helpful comments and suggestions.

REFERENCES

[1] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 133–153, 2008.

[2] S. Toulmin, R. Rieke, and A. Janik, *An Introduction to Reasoning*. Macmillan, 1979.

[3] T. P. Kelly, "Arguing Safety - A Systematic Approach to Safety Case Management," Ph.D. dissertation, University of York, 1998.

[4] B. Burgemeestre, J. Hulstijn, and Y.-H. Tan, "Value-Based Argumentation for Justifying Compliance," in *DEON'2010*. Springer, 2010, pp. 214–228.

[5] R. A. Weaver, "The Safety of Software: Constructing and Assuring Arguments," Ph.D. dissertation, University of York, September 2003.

[6] S. Drimer, S. J. Murdoch, and R. Anderson, "Thinking Inside the Box: System-Level Failures of Tamper Proofing ," in *SP'2008*. IEEE Press, May 2008, pp. 281–295.

[7] Visa, "Testing and Approval," Website, 2011, https://partnernetwork.visa.com/vpn/global/category.do?categoryId=103&documentId=501&userRegion=1, last visited Feb 2011.

[8] "Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 3, CCMB-2007-09-001, CCMB-2007-09-002 and CCMB-2007-09-003," July 2009.

[9] S. Drimer, S. J. Murdoch, and R. Anderson, "Failures of Tamper-Proofing in PIN Entry Devices," *IEEE Security and Privacy*, vol. 7, pp. 39–45, November 2009.

[10] M. Jackson, *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley/ACM Press, 2001.

[11] S. E. Newman and C. C. Marshall, "Pushing Toulmin Too Far: Learning from an Argument Representation Scheme," Xerox PARC, Tech. Rep. SSL-92-45, 1991.

[12] S. Barnum, *Common Attack Pattern Enumeration and Classification (CAPEC) Schema Description*, Copyright Cigital Inc., commissioned by the U.S. Department of Homeland Security, January 2008, http://capec.mitre.org/documents/documentation/CAPEC_Schema_Description_v1.3.pdf, Version 1.3.

[13] CWE Team, "CWE Schema Documentation," Online by The MITRE Corporation, December 2010, http://cwe.mitre.org/documents/schema/index.html, Version 4.4.2.

[14] The-Card-Payment-Group, "PIN Entry Device Protection Profile," Common Criteria Portal, Jul 2003, www.commoncriteriaportal.org/files/ppfiles/PED_PPv1_37.pdf, last visited Jul 2010.

[15] Mastercard International Incorporated, "Payment Card Industry POS PIN Entry Device Security Requirements," m2m Group website, http://www.m2mgroup.ma/livresetdocs/security%20risk.htm, last visited Feb 2011, October 2004, version 7 1.0, Revised March 2005.

[16] R. Anderson, "Failures on Fraud," *Speed*, vol. 3, no. 2, pp. 6–7, September 2008.

[17] R. A. Carter, A. I. Antón, L. A. Williams, and A. Dagnino, "Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model," in *RE'01*, 2001, pp. 94–101.

[18] J. Chisan and D. Damian, "Exploring the role of requirements engineering in improving risk management," in *RE'05*, 2005, pp. 481–482.

[19] A. V. Miranskyy, N. H. Madhavji, M. Davison, and M. Reesor, "Modelling Assumptions and Requirements in the Context of Project Risk," in *RE'05*, 2005, pp. 471–472.

[20] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering Journal*, vol. 10, no. 1, pp. 34–44, 2005.

[21] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," in *ACSAC'99*. IEEE Press, 1999, pp. 55–64.

[22] B. Schneier, "Attack Trees: Modeling Security Threats," *Dr. Dobb's Journal*, December 1999.

[23] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *RE'04*. IEEE Computer Society, 2004, pp. 354–355.

[24] A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," in *ICSE '04*. IEEE Press, 2004, pp. 148–157.

[25] I. A. Tøndel, J. Jensen, and L. Røstad, "Combining Misuse Cases with Attack Trees and Security Activity Models," in *ARES'2010*. IEEE Press, 2010, pp. 438–445.

[26] P. Graydon and J. Knight, "Success Arguments: Establishing Confidence in Software Development," University of Virginia, Tech. Rep. CS-2008-10, July 2008.

[27] S. B. Shum and N. Hammond, "Argumentation-based Design Rationale: What Use at What Cost?" *Int. Journal of Human-Computer Studies*, vol. 40, no. 4, pp. 603–652, 1994.

[28] D. N. Walton, *Argumentation Schemes for Presumptive Reasoning*. Mahwah NJ, USA: Lawrence Erlbaum Associates, 1996.

[29] K. Atkinson, T. Bench-Capon, and P. McBurney, "Justifying Practical Reasoning," in *CMNA'04*, 2004, pp. 87–90.

[30] P. Baroni, F. Cerutti, M. Giacomin, and G. Guida, "An Argumentation-Based Approach to Modeling Decision Support Contexts with What-If Capabilities," in *AAAI Fall Symposium. Technical Report SS-09-06*. AAAI Press, 2009, pp. 2–7.

[31] H. Lipson and C. Weinstock, "Evidence of Assurance: Laying the Foundation for a Credible Security Case," May 2008, department of Homeland Security; online: https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance/973-BSI.html, last visited Feb 2011.

[32] F. den Braber, I. Hogganvik, M. S. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps - a guided tour to the CORAS method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[33] G. Brndeland, H. E. Dahl, I. Engan, and K. Stölen, "Using Dependent CORAS Diagrams to Analyse Mutual Dependency," in *CRITIS'2007*, ser. LNCS 5141. Springer Press, 2008, pp. 135–148.

[34] Walton-on-Thames: Insight Consulting, "CRAMM User Guide," July 2005, risk Analysis and Management Method, Version 5.1.

[35] ISO/IEC-27001/27005, "Information technology. Security techniques. (27001) Information security management systems; (27005) Information security risk management." 2008.

[36] AS/NZS-4360:2004, "Australian/New Zealand Standards, Risk Management," Sydney, NSW, 2004.

[37] P. H. Engebretson and J. J. Pauli, "Leveraging Parent Mitigations and Threats for CAPEC-Driven Hierarchies," in *ITNG'09*. IEEE Press, 2009, pp. 344–349.

[38] Y. Yu, T. T. Tun, A. Tedeschi, V. N. L. Franqueira, and B. Nuseibeh, "Openargue: Supporting argumentation to evolve secure software systems," in *RE'11*. IEEE press, 2011.

[39] "Common Weakness Scoring System (CWSS)," Online: http://cwe.mitre.org/cwss/#vectors, 2011, version 0.2, 14 February 2011.

[40] V. N. L. Franqueira, S. Houmb, and M. Daneva, "Using Real Option Thinking to Improve Decision Making in Security Investment," in *IS'2010 (OTM Conferences)*, ser. LNCS. Springer Press, 2010, pp. 619–638.

[41] R. Haenni, B. Anrig, J. Kohlas, and N. Lehmann, "A Survey on Probabilistic Argumentation," in *ECSQARU'01*, 2001, pp. 19–25.

APPENDIX D

Managing Evolution by Orchestrating Requirements and Testing Engineering Processes

Federica Paci[1], Fabio Massacci[1], Fabrice Bouquet[2], and Stephane Debricon[2]

[1] DISI,University of Trento
{Fabio.Massacci,Federica.Paci}@unitn.it
[2] Laboratoire d'Informatique, Université de France-Comté
{fabrice.bouquet,stephane.debricon}@lifc.univ-fcomte.fr

Abstract. Change management and change propagation across the various models of the system (such as requirements, design and testing models) are well-known problems in software engineering. For such problems a number of solutions have been proposed that are usually based on the idea of integrated model repositories where traceability links are mantained and automatically triggered.
We propose to manage the mutual evolution of requirements models and tests models by orchestrating processes based on a minimal shared interface. Thus, requirement and testing engineers must only have a basic knowledge about the "other" domain, share a minimal set of concepts and can follow their "own" respective processes. The processes are orchestrated in the sense that when a change affects a concept of the interface, the change is propagated to the other domain. We illustrate the approach using the evolution of the GlobalPlatform® standard.

1 Introduction

Change management is a well known problem in software engineering and in particular the change propagation across the various models of the system (such as requirements, design and testing models). For such problem a number of solutions have been proposed that are usually based on the idea of integrated model repositories where traceability links are mantained and automatically triggered.

For particularly complex and security critical systems as multi-application smart cards [1] such sharing may simply not be possible. Here is a concrete example: test engineer in charge of certifying that the card is secure might not have access to system code (and the relative design models) simply because he is part of a third-party company to which the certification has been outsourced. Still, the test engineer must cooperate in the global design process to show that requirements are achieved. Fig. 1 shows an example of global security engineering process where the activities are labeled with references to clauses of a national security standard.

Thus, it is important for test and requirement engineer to interact directly and bypass completely the system designer. The cooperation between Test Analyst (the professional name for model-driven test engineer) and Business Analyst (the requirement engineer) is crucial. The Test Analyst is responsible for the quality of the test repository in terms of coverage of requirements and detection of defects. In the other direction, the Test Analyst interacts with the Tester and Test Automation Engineer to facilitate

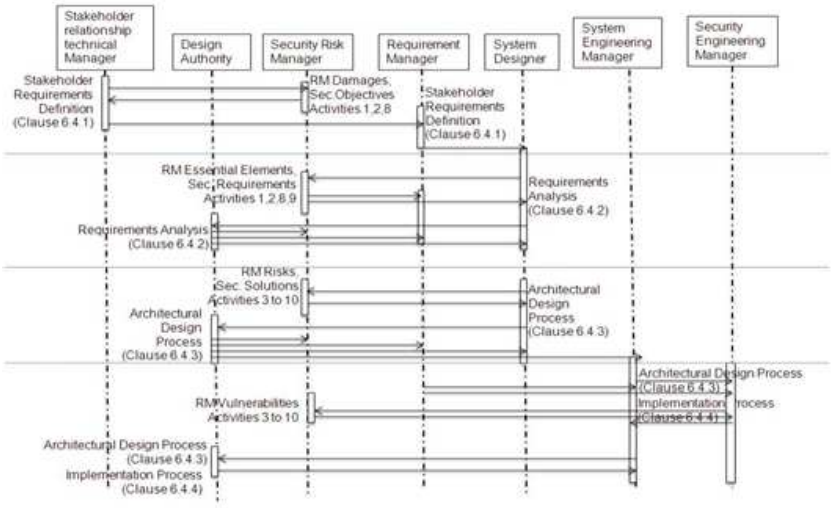


Fig. 1. Security Engineering Process

manual test execution or test automation (implementation of keywords). This interaction process is highly iterative. The model-driven testing process involves three main roles:

- The Business Analyst is the reference person for the System-Under-Test (SUT) requirements, business processes and business needs, and he/she communicates with the Test Analyst to clarify the specifications and testing needs.
- The Test Analyst interacts with the Business Analysts and system experts regarding the requirements to be covered, and then designs test generation models. He then uses test generation tools to automatically generate tests and to produce a repository of test suites that will satisfy the project test objectives.
- A Tester is responsible for (manual) test execution on the basis of the test repository while a Test Automation Engineer is responsible for connecting the generated tests to the system under test so that the tests can be executed automatically. The input for both Tester and Test Automation Engineer is the test repository generated automatically by the test analyst from the test generation models.

For this process to work smoothly in presence of changes we need to orchestrate the work of the requirement engineer with the work of the testing engineer. In many cases, requirements evolution can have impact on a confined part of the system. In such cases it would be beneficial to clearly identify only those parts of the system that have been affected by the evolution and that need to be re-tested for compliance with requirements. In this way re-running all test cases is avoided because it is possible to identify which new test case need to be added to the test suite and which test cases can be discarded as obsolete.

1.1 The Contribution of this Paper

We propose a framework for managing the impact of changes happening at requirement level on testing generation process and viceversa. The key features of the framework are *model-based traceability* by *orchestration* and *separation of concerns* between the requirement and the testing domain. Separation of concern allows the requirement engineer to have very little knowledge about the test (process, modeling or generation) domain, and similarly for the test engineer. They only share a minimal set of concepts which is the *interface* between the requirement and the testing frameworks. Moreover, both engineers simply need to follow their respective processes (i.e., requirement engineering and testing generation process) separately. The processes are *orchestrated* in the sense that when a change affects a concept of the interface, the change is propagated to the other domain. The interface also supports traceability between the requirement and test models through mapping of concepts in the two domains.

For sake of concreteness, we instantiate the requirement framework to SI* [11] and the test framework to SeTGaM [9] . However, our approach is independent from the specific requirement and testing frameworks that are adopted, and can thus be applied to other competing instantiations if these have mapping concepts similar to the ones we propose in Section 6.

In the next section we introduce the evolution of the GlobalPlatform standard for multi-application smard cards that will be our running example. Then we describe the overall orchestrated process (§3) and how changes are managed at the requirements level (§4) and at the level of model-based testing (§5). Sec.6 illustrates the conceptual interface while Sec.7 presents the application of the process to the case study. Finally we discuss related works in Sec. 8 and conclude the paper in Sec 9.

2 GlobalPlatform Evolution

The most popular solution for smart cards now is *GlobalPlatform* (GP) [1] on top of Java Card [19]. Loosely speaking GP is a set of card management services such as loading, enabling, or removing applications. The GP specification describes the card life cycle and the GP components that are authorized to perform a transition in the life cycle: Security Domains and Applications. Security Domains act as the on-card representatives of off-card authorities such as the Card Issuer or Application Providers.

The card life cycle begins with the state OP_READY. Then the card can be set to the states INITIALIZED, SECURED, CARD_LOCKED and TERMINATED that is the state where the card cannot longer be used. During the evolution of card life cycle between versions 2.1.1 and 2.2 of GP specification as illustrated in Figure 2 a number of changes took place giving authority to perform transitions to different stakeholders. In GP-2.1.1 a privileged application can terminate the card from any state, except CARD_LOCKED. Additionally, a privileged application can lock the card by changing card state from SECURED to CARD_LOCKED. However, only the issuer of the card can move it to TERMINATED state. A security domain is a special kind of privileged application, and therefore, has exactly the same behavior of privileged application in terms of card lifecycle management. In GP-2.2 two main changes with respect to GP-2.1.1 are introduced: a) a privileged application can terminate the card from any state if

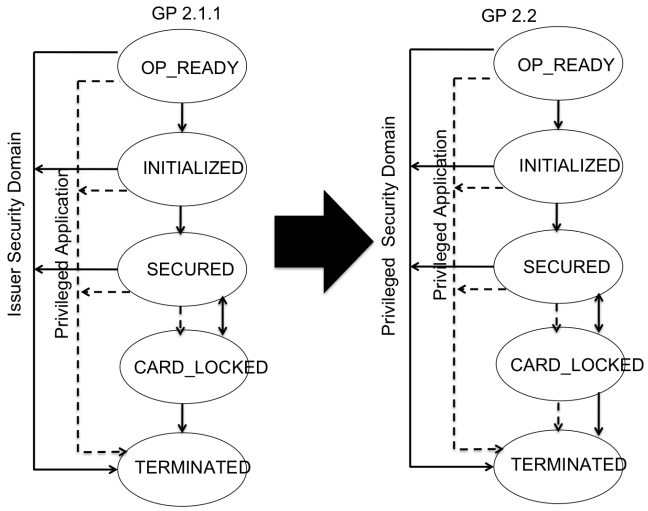


Fig. 2. Card Life Cycle in GP-2.1.1 and GP-2.2

the application has appropriate privileges; b) any privileged security domain can trigger all card life cycle transitions while in GP-2.1.1 only the issuer security domain can do that.

3 Orchestrated Change Management Process

The orchestration of the test and the requirements engineering processes is based on the principle of *separation of concern*: the two processes should be understood as separate processes with their own iterations, activities and techniques for managing change.

The UML activity diagram of Fig. 3 gives a high-level overview of the orchestrated process. The diagram is divided into three partitions to distinguish between the activities and objects under the control of users, requirement engineers, and test engineers. A user is typically the client commissioning the testing and may be the owner of the SUT. In the diagram, the diamonds specifies branching of the sequence of activities. When there is no guard condition on the branching (specified by the notes with Boolean expressions), the process proceeds along one or both of the branches. This gives a wide flexibility on how the overall process may be conducted.

Interactions are triggered by the change request from the user. When this change request is passed to one or both the test engineer and the requirement engineering, there are a number of iterations where the changes propagate back and forth between the two until a stable state (equilibrium) is reached and the results can be passed back to the user. At the model level, these concrete triggers are changes in a small set of concepts that works as the interface between the requirement engineering process and the test engineering process. We will describe these concepts later in Section 6.

To illustrate the process, we start from changes in the requirement domain:

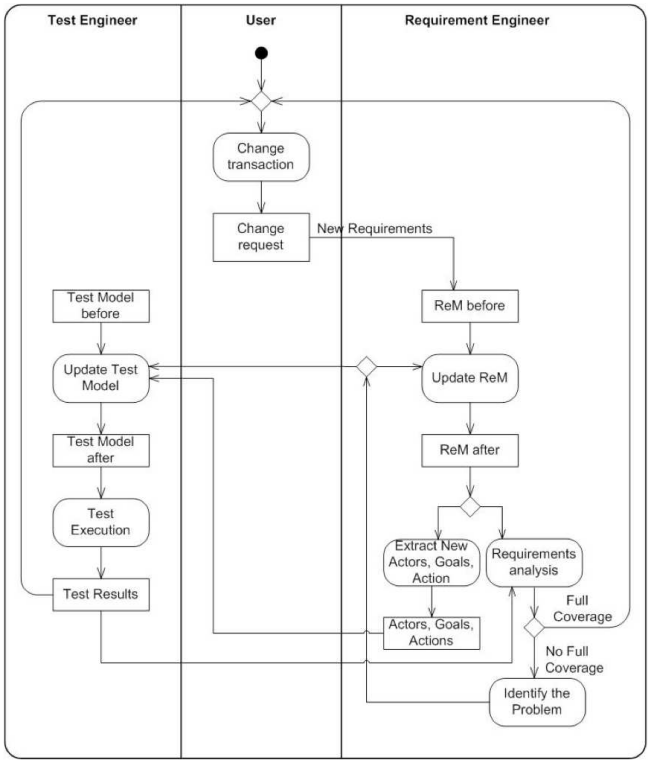


Fig. 3. Integrated Change Management Process

Update ReM. The requirement engineer uses the previous requirement model (ReM-before) and the change request to update the model, producing ReM-after.

Extract New Actors, Goals, Actions. Based on the ReM-after, new actors, goals and processes are extracted if relevant and provided to the test engineer.

Update Test Model. Receiving the extracted actors, goals and processes the test engineer based on the traceability links between the ReM and the test model (TeM), identify the part of the TeM that are affected by the changes in the ReM. The test engineer thus updates the TeM and the test suite for the updated TeM (See Sec.5 for the test suite generation).

Test Execution.Then, the test engineer executes the new test suite. The test engineer returns the test results to the requirement engineer in a suitable table. The table shows for each test case in the test suite the number of times the test has been executed, the status of the test after evolution, the TeM element and the requirements/goals covered by the test case, and the test result.

Requirement Analysis. The requirement engineer evaluates the matrix for each requirement covered by the test and translates the test results into a level of achievement (partial satisfaction/denial or full satisfaction/denial) for the low level requirements. Once the requirement engineer gets the achievement levels for low level re-

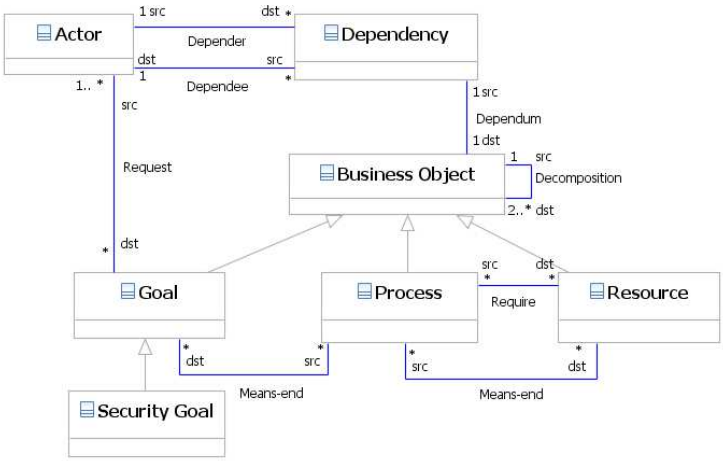


Fig. 4. SI* conceptual model

quirements, he can run the requirement analysis to determine the level of achievement also for top-level requirements.

Identify the problem. If some of the requirements are not fulfilled, the requirement engineer must identify the problem.

1. If there is a problem with the ReM, the requirement engineer must backtrack and search for an alternative way of updating the ReM when considering the change request that was initially passed from the user.

b If there is a problem with testing, the test engineer must determine whether there is the need to generate new test cases or not.

4 Change management for evolving requirements

For the change management process in the requirement domain we consider here the SI* requirement model [11] based on the Tropos methodology. As illustrated in Fig. 4, we consider the concept of *goal* and a subset of SI* relations, namely AND/OR *decomposition*, *means-end*, *require*, *request*, and *dependency* relations.

The requirement analysis process consists of five steps:

1. Identify relevant stakeholders, modeled as *actor* (circle) and its structure.
2. Capture and refine actors'requirements as *goal* (rounded rectangle).
3. Define means to achieve their goals- i.e., *process* (hexagon) or *resource* (rectangle).
4. Model strategic dependencies between actors in fulfilling/executing/providing some goals/processes/resources.
5. Model specific security and risk related aspects such as introducing *security goals*, which are goals protecting assets and that can be identified as result of risk analysis [24] or analyzing trust relations and delegation of permissions among actors [11].

The requirement analysis is an iterative process that aims at refining the stakeholders' goals until all high-level goals are achieved. Different reasoning techniques for achievement can be applied based on goals [25] or arguments and logic-programs [12], or Datalog and logic programs [11], or a combination of qualitative and quantitative criteria [2]. Since testing may not provide full fledged evidence we may use the approach by Asnar et al [2] where we use the keyword SAT to denote that the evidence is in favor of the achievement of the the goal and DEN to denote that the evidence is against it.

The evolution of a requirements model can be triggered by a *change request* that can be placed by stakeholders, or it can be a reaction to a previous change, or caused by external circumstances and merely observed. By means of *evolution rules*, it is possible to automatically detect changes in the requirement model and it is possible to define reactions to changes. Evolution rules are defined in conformance with the Event - Condition - Action semantics: basically, an *Event* captures a dynamic change in the requirement model, while *Condition* identifies the static context where this change happened. An *Action* is a list of operations that constitute the reaction to that event.

An example of evolution that can be detected by an evolution rule is the addition of an actor to the requirement model and the delegation of a security goal by the actor to another actor which is not trusted. The Event and Condition part of the rule can be mapped on the addition of the new actor and the missing trust relationship while the Action part can specify appropriate reaction ranging from logging the event to automatic intervention like creating the missing trust relationship.

Evolution rules can also be used to propagate changes in requirement models to other artefacts e.g risk models and test models (as in our case).

5 Change management for evolving tests

As testing generation process we consider SeTGaM [10]. The model-based testing generation process starts by the design of the test model by the test architect: the model should describe the expected behaviour of the system under test (SUT). Then, the test model is used to generate the test cases and the coverage matrix, relating the tests with the covered model elements. The tests are then exported, or published, in a test repository and then executed. After the test execution, test results and metrics are provided.

The test model consists of three different types of UML diagrams (Fig. 5). First, a class diagram describes the data model, namely the set of classes that represent the entities of the system, with their attributes and operations. Second, an object diagram provides a given instantiation of the class diagram together with the test data (i.e. the objects) that will be used as parameters for the operations composing the tests. Finally, the behavior of the system is described by two (complementary) means: a statechart diagram, and/or OCL constraints associated with the operations of the class diagram. The test coverage of system requirements and test objectives is achieved byusing the tags @REM and @AIM to annotate the OCL code.

When an evolution occurs, the status of the test changes depending on the impact of the evolution on the model elements covered by the test case. Evolution of status is defined by considering two versions of the test model, M and M', in which addition, modification or deletion of model elements (operations, behaviors, transitions,

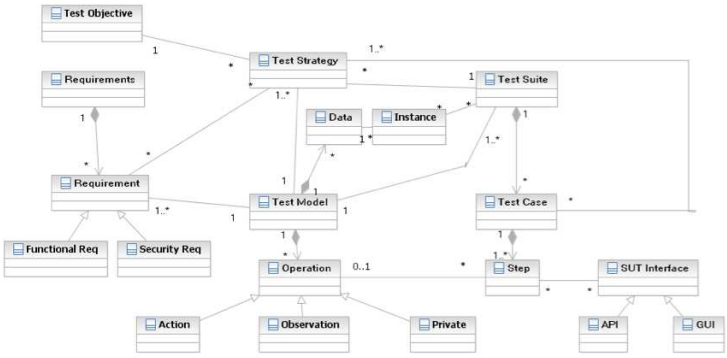


Fig. 5. Basic testing concepts

Legend:

- *requirement*: statement about what the system should do
- *test model*: dedicated model for capturing the expected SUT behavior (Class diagram, State machine)
- *test case*: a finite sequence of test steps
- *test intention*: user's view of testing needs
- *test suite*: a finite set of test cases
- *test script*: executable version of a test case
- *test step*: operation's call or verdict computation
- *test objective*: high level test intention.

etc.) have been performed. Test may have a status *new* in case of a newly generated test for a newly introduced target.

If none of the model elements covered by the test is impacted, the test is run as is on the new version of the model M', without modifying the test sequence. The test is thus said to be *reusable*. More precisely, there are two cases: *unimpacted* and *re-executed*. A test is unimpacted if the test sequence is identical to its previous version, and the covered requirements still exist. The test is re-executed if it covers impacted model elements, but it can still be animated on the new version of the model without any modification.

If a test covers model elements impacted by the evolution from M to M', and if the test cannot be animated on M' the test becomes *obsolete*. There are two cases: either the target represents deleted model elements, and thus the test does not make any sense on M' and it is said to be *outdated*, or, the test fails when animated on model M' (e.g. due to a modification of the system behaviour), it is then *failed*. When the test case operations can be animated but produce different outputs, a new version of the test is created in which the expected outputs (i.e. the oracle) are updated w.r.t. M'. In this case the tests have the status *updated*. When the test case operations can not be animated as is in the first version of the test, a new operation sequence has to be computed to cover the test target. In the latter case, tests have status *adapted*.

To determine the status of a test when evolution takes place, the SeTGaM approach relies on dependency analysis that is performed to compute the differences between the models, and their impacts on test cases. We have four different classification suites.

- *evolution test suite* contains tests classified as new and adapted;

Table 1. Conceptual Interface

Requirement Concept	Testing Concept	Kind of Interaction
Requirement	Requirement	Shared concept
Goal	Test Model (State Machine, OCL code)	Mapped concept
Process	Test Model (State Machine, OCL code)	Mapped concept
Actor	SUT	Mapped concept
Achievement Level	Test result & status	Mapped concept

- *regression test suite* contains tests classified as unimpacted and re-executed;
- *stagnation test suite* contains tests classified as outdated and failed.
- *deletion test suite* contains tests, that come from the stagnation test suite from the previous version of the model.

6 Change Management Conceptual Interface

The orchestration of the requirements engineering process and the test generation process is based on the identification of a set of concepts that are shared or mappable in the two domains: a *shared concept* is a concept that has the same semantics in both domains while a *mappable concept* is a concept that is related to one in the other domain. Tab. 1 illustrates the conceptual interface. When a concept is changed in a model then a corresponding change request is issued to the other model.

We identify one shared concepts that is *Requirement*. A Requirement in both domains represents a statement by a stakeholder about what the system should do. The concepts of *Actor*, *Goal*, *Process* are mapped on the Test Model. In particular, the concept of Actor is used to identify the system under test (SUT). The concepts of Goal and Process are used by the testing engineer to build the different types of Test Models. The goals and processes in the Requirement Model are identified by a unique name that is used to annotate the State Machine of the Test Model and the OCL code in order to achieve traceability between the Requirement Model and the Test Model.

Mapping of a test case's result and status to a requirement achievement level allows the requirement engineer to quantify the requirement *coverage* after evolution. This correspondence is reported in Table 2: if the status of a test case after evolution is *new*, *adapted* or *updated*, and the test result is *pass* the requirement covered by the test case is fulfilled while it is denied (i.e. we have evidence that has not been achieved) if the test result is *fail*. A subtle case is present when a test case is part of the stagnation suite (i.e. *obsolete*) and the test result is *fail*. Here the test covers requirements that have been deleted from the model and thus the corresponding behavior should no longer be present (for example a withdrawn authorization) so failing the test shows that the unwanted behavior is no longer present.

We also consider *completion* indicators for the change propagation process which indicates whether all changes in the requirement model have been propagated to the test model. Table 3 summarizes the mapping between Goal and Process in the requirement model and the Test Model element. In a nutshell we say that the change propagation process has been completed if:

Table 2. Requirements Coverage

Test Classification	Test Status	Test Result	Achievement Level
Evolution	New, Adapted,Updated	Pass	Fulfill
Regression	Unimpacted, Re-Executable	Pass	Fulfill
Evolution	New, Adapted,Updated	Fail	Deny
Regression	Unimpacted, Re-Executable	Fail	Deny
Stagnation	Outdated, Failed	Pass	Deny
Stagnation	Outdated, Failed	Fail	Fulfill

Table 3. Completion of Change Propagation

Change in ReM Model	Test Status	Test Suite
New Element (Goal, Process)	New	Evolution
Modified Element (Goal, Process)	Adapted	Evolution
Model Element Not Impacted	Re-Executable	Regression
Deleted Element	Obsolete	Stagnation

- for each new or modified model element in the ReM model a new test case and an adapted are added to the evolution test suite,
- for each model element not impacted by evolution there is a re-executable test case in the regression test suite,
- for each model element deleted form the model there is an obsolete test case in the stagnation test suite.

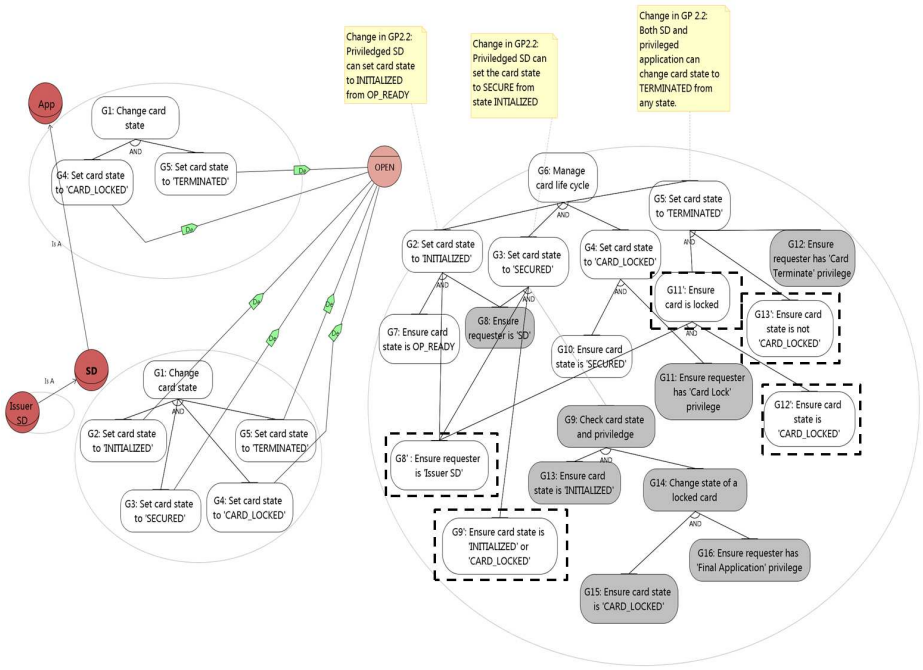
7 Application to Case Study

We first illustrate how a change from the GP-2.1.1 to the GP-2.2 requirement model is propagated to the test model and thus how the test suite for the GP-2.2 test model is generated. Then, we show how the test classification for the test model of GP-2.2 can be used to evaluate the *completion* of the change propagation process.

Fig. 6 shows the SI* model for GP-2.1.1 and 2.2. The main actors are Global Platform Environment (OPEN), Privileged application (App), Privileged Security Domain (SD), and Issuer Security Domain (ISD). We only focus on the card lifecycle transition to TERMINATED state that is the one impacted by the evolution. This transition is represented by the goal G5: in the GP-2.1.1 model the goal G5 is AND decomposed in two subgoals G13 and G11' the latter further decomposed into subgoals in goals G8' and G12'; in the GP-2.2 model the goal G5 has only G12 as subgoal (labeled in grey).

Fig. 7 represents the test model for GP-2.1.1 and GP-2.2: transitions *SetOpNopSD* and *SetInNopSD* (dotted arrows) are removed in GP-2.2 because they are associated with the deleted goal G9'. The transitions *SetStatusNoApp* and *SetStatusNoApp* (bolded arrows) between the states *Card_Locked* to *Terminated* are added to the test model because in GP-2.2 requirement model the decomposition of G5 goal is changed.

The traceability link between the goals in Fig. 6 and the transitions in the test model of Fig. 7 is illustrated in Fig. 8 representing the dynamic behavior of the transition *setStatusApp*. In order to trace the transition to goals G5 and G12 , the OCL code is



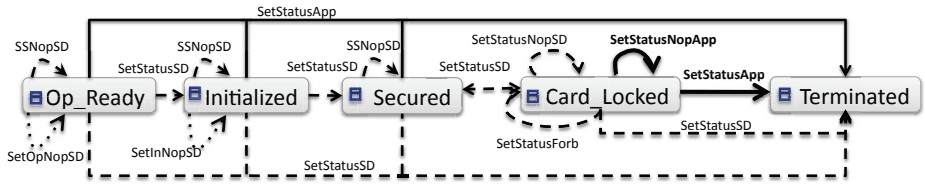
Legend: Goals surrounded by dashed rectangles correspond to requirements that belong only to G-2.1.1 model, goals in grey are new requirements related to the card life cycle introduced in version 2.2, and goals in white are goals corresponding to requirements that are present in both versions.

Fig. 6. Requirement Model for GP specs 2.1.1 and 2.2

Table 4. Test Suite for GP-2.1.1

Transition Covered	Test	Requirement
SetStatusForb	$Test_1$	$G_6, G_{11'}$
SetStatusNopSD	$Test_2$	$G_5, G_{8'}, G_{11'}$
SetStatusSD	$Test_3$	$G_5, G_{8'}, G_{11'}, G_{12'}$

annotated with the tags @REM G5 and @REM G12 referring the goals G5 and G12. Based on the traceability link between goals and test model transitions we can generate the test suites for GP-2.1.1 and GP-2.2 that are illustrated respectively in Tab. 4 and Tab. 5. The tables only focus on the test cases for transitions from *Card_Locked* to *Terminated* states: *SetStatusNopSD* and *SetStatusNopApp* that correspond to *setStatus* command performed by a Security Domain and Application with no Terminate privilege, *SetStatusSD* and *SetStatusApp* correspond to *setStatus* command performed by a Security Domain and Application with Terminate privilege and *SetStatusForb* corresponding to a Security Domain and Application with no Card Locked privilege. For example, since a new goal G12 has been added to the SI* model for GP-2.2 two new test cases *Test 3* and *Test 5* covering G12 and its top goal G5 have been added to the test suite.



Legend: Dotted arrows correspond to transitions that were part of GP 2.1.1 test model and has been removed in the model of GP 2.2, bolded arrows represent new transitions, dashed arrows correspond to modified transitions, while full arrows correspond to transitions not impacted by evolution of requirements.

Fig. 7. Test Model for GP specifications 2.1.1 and 2.2

Guard - setStatusCardLockedToTerminated_privilegedApp

```
1 /*APDU: SetStatus GP2.2
2 Type: Guard of transition
3
4 Current state is CARD_LOCKED ,
5 Application = not an SD but with cardTerminate privilege */
6 (
7     self.lcs->exists(lc : LogicalChannel |
8         IN_lcNumber = lc.number and
9         IN_claSMLevel = lc.secureChannelSession.secureMessagingLevel and
10        /**@REQ: G12*/ /*application with cardTerminate privilege*/
11        lc.selectedApp.privileges.cardTerminate = true and
12        lc.selectedApp.privileges.securityDomain = false
13    ) and
14    IN_option = ALL_SET_STATUS_OPTIONS::CARD and
15    self.state = ALL_STATES::CARD_LOCKED and
16    /**@REQ: G5*/ /*new state is TERMINATED*/
17    IN_state = ALL_STATES::TERMINATED
18) = true
```

Legend: OCL code for the transition *setStatusApp* from CARD-LOCKED to TERMINATED requested by an Application with cardTerminate privilege. The code is annotated with the identifiers of the goals G5 (@REM G5) and G12 (@REM G12) covered by the test cases *Test 3* and *Test 5*.

Fig. 8. OCL code for SetStatus APDU command setting card state to TERMINATED

With respect to the completion of the change propagation process, we can see that the changes in the card life cycle related to the state TERMINATED has been propagated from the requirement model to the test model: two new test cases *Test 4* and *Test 5* and two updated test cases *Test 2* and *Test 3* corresponding to G5 and its subgoals has been included in the evolution test suite. *Test 4* and *Test 5* correspond to an Application executing *setStatus* command without and with Terminate privilege respectively, while *Test 2* and *Test 3* are related to the execution of the *setStatus* command performed by a Security Domain without and with Terminate privilege.

8 Related Works

Change management is well known for being a difficult and costly process. However, only some requirement engineering proposals provide support for handling change propagation and for change impact analysis. Goal-oriented approaches such as KAOS,

Table 5. Test Suite for GP-2.2

Transition Covered	Test	Requirement	Status
SetStatusForb	$Test_1$	G_6, G_{11}	Re-executed
SetStatusNopSD	$Test_2$	G_5, G_8, G_{11}	Updated
SetStatusSD	$Test_3$	$G_5, G_8, G_{12}, G_{15}, G_{16}$	Updated
SetStatusNopApp	$Test_4$	G_5, G_{11}, G_{15}	New
SetStatusApp	$Test_5$	$G_5, G_{12}, G_{15}, G_{16}$	New

Secure Tropos, and Secure i* [24, 11, 18] provide good support for change propagation because they are based on goal models which explicitly show relationships and dependencies between goals, and also support the modeling and the analysis of dependencies between agents. Tanabe et al. [22] propose an approach to requirements change management that supports version control for goal graphs and impact analysis of adding and deleting goals.

UMLsec [15] is a model-based approach to security engineering which supports change impact analysis by using model-checking and theorem proving techniques.

Problem-oriented approaches also support change impact analysis to some extent. Haley et al. [13] use argument satisfaction as a way of verifying that a specification satisfies a requirement in a given context. Lin et al. [16] do not provide explicit support for change impact analysis, but this can be achieved by using problem analysis when new security problems are identified.

Chechik et al. [4] propose a model-based approach to propagate changes between requirements and design models that utilize the relationship between the models to automatically propagate changes. Hassine et al. [14] present an approach to change impact analysis that applies both slicing and dependency analysis at the Use Case Map specification level to identify the potential impact of requirement changes on the overall system. Lin et al. [17] propose capturing requirement changes as a series of atomic changes in specifications and using algorithms to relate changes in requirements to corresponding changes in specifications.

With respect to change management in test engineering, several works about regression testing have been proposed.There are two kinds of regression testing: *code-based* regression testing and *specification-based* regression testing.

Code-based testing is limited to unit testing, and is mainly applied to concurrent programs ([7]). At program level, in [6], the authors describe how to select a tests' subset to be used for regression testing. This subset is defined by using data coverage of the test w.r.t. the changes that occurred in a program.

In the specification-based regression testing field, a variety of techniques can be found, based on various selection criteria, such as requirement coverage [5]. In [23] the authors use EFSM models for safe regression technique based on dependence analysis. They select test cases and compute the regression test suite by identifying three types of elementary modifications applicable to a machine (addition, deletion, modification of a transition). Our approach is grounded on these principles, but improves them by keeping the test history. In addition, we consider three test suites fulfilling different purposes. In [8] the authors propose a methodology to identify impacted part of the model. A list of all depending operations is created for each operation modification. They identify all parts of dynamic UML diagrams in which the behaviour of this operation can

be found. This approach can be seen as a variation of the approach proposed here, that does necessarily consider statecharts diagrams. The authors present in [21] a regression testing approach based on Object Method Directed Acyclic Graph (OMDAG) using class diagrams, sequence diagrams and OCL code. They consider that a change in a path of the OMDAG affects one or more test cases associated to the path. They classify changes as NEWSET, MODSET and DELSET, which can be identified as the elementary modifications we consider. In [20] a model-based selective regression technique is described, based on UML sequence diagrams and OCL code used to describe the system's behavior. In [3] the author describe a regression testing method using UML class, sequence diagrams and use case diagrams. Changes in actions are collected by observing sequence diagrams, while changes in the variables, operations (OCL), relationships and classes are collected by comparing class diagrams.

9 Conclusion and Future Works

In this paper we have proposed a novel framework for managing the mutual impact of changes happening at requirement and testing models level. The key features of the framework are *model-based traceability* by *orchestration* and *separation of concerns* between the requirement and the testing domain. Separation of concern allows the requirement engineer and the test engineer to cooperate by only sharing a minimal set of concepts which is the *interface* between their respective change management processes. The processes are *orchestrated* in the sense that when a change affects a concept of the interface, the change is propagated to the other domain.

We are planning to extend the framework in order to support the semi-automatic propagation of changes between the requirement and testing models by means of incremental transformation rules.

Acknowledgment

This work was partly supported by the project EU-FP7-FET-IP-SecureChange (`http://www.securechange.eu`)

References

1. Global platform specification. `http://www.globalplatform.org`, May, 2011. v.2.1.1 available in March'03 and v.2.2 available in March'06.
2. Y. Asnar, P. Giorgini, and J. Mylopoulos. Goal-driven risk assessment in requirements engineering. *REJ*, pages 1–16, 2011.
3. L. Briand, Y. Labiche, and G.Soccar. Automating impact analysis and regression test selection based on uml designs. In *Proc. of ICSM '02*, page 252, 2002.
4. M. Chechik, W. Lai, S. Nejati, J. Cabot, Z. Diskin, S. Easterbrook, M. Sabetzadeh, and R. Salay. Relationship-based change propagation: A case study. In *Proc. of MISE'09*, pages 7–12. IEEE Press, 2009.
5. P. K. Chittimalli and M. J. Harrold. Regression test selection on system requirements. In *Proc. of the 1st India Soft. Eng. Conf. (ISEC'08)*, pages 87–96. ACM, 2008.

6. P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. *TSE*, 35(4):452–469, 2009.
7. I. S. Chung, H. S. Kim, H. S. Bae, Y. R. Kwon, and D. G. Lee. Testing of concurrent programs after specification changes. In *Proc. of ICSM '99*, page 199, 1999.
8. D. Deng, P. C. Y. Sheu, T. Wang, and A. K. Onoma. Model-based testing and maintenance. In *Proc. of ISMSE'04*, pages 278–285. IEEE Press, 2004.
9. E. Fourneret, F. Bouquet, F. Dadeau, and S. Debricon. Selective test generation method for evolving critical systems. In *REGRESSION'11*. IEEE Press, 2011.
10. E. Fourneret, F. Bouquet, F. Dadeau, and S. Debricon. Selective test generation method for evolving critical systems. In *Proc. of ICST'11*, 2011.
11. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *IJIS*, 5(4):257–274, 2006.
12. R. Haenni and N. Lehmann. Probabilistic argumentation systems: a new perspective on the dempster-shafer theory. *International Journal of Intelligent Systems*, 18(1):93–106, 2003.
13. C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *TSE*, 34:133–153, 2008.
14. J. Hassine, J. Rilling, and J. Hewitt. Change impact analysis for requirement evolution using use case maps. In *Proc. of the 8th Int. Workshop on Principles of Soft. Evolution*, pages 81–90. IEEE Press, 2005.
15. J. Jurjens. *UMLsec: Extending UML for Secure Systems Development*. Springer Verlag, 2002.
16. L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett. Introducing abuse frames for analysing security requirements. In *Proc. of RE'03*, pages 371–372, 2003.
17. L. Lin, S. J. Prowell, and J. H. Poore. The impact of requirements changes on specifications and state machines. *Softw. Pract. Exper.*, 39:573–610, April 2009.
18. L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In *Proc. of RE'03*, pages 151–161, 2003.
19. S. Microsystems. Runtime environment specification. Java Card™ platform, Connected edition. Specification 3.0, 2008.
20. L. Naslavsky, H. Ziv, and D. J. Richardson. Mbsrt2: Model-based selective regression testing with traceability. In *Proc. of ICST'10*, pages 89–98. IEEE Press, 2010.
21. O. Pilskalns, G. Uyan, and A. Andrews. Regression testing uml designs. In *Proc. of ICSM'06*, pages 254–264, 2006.
22. D. Tanabe, K. Uno, K. Akemine, T. Yoshikawa, H. Kaiya, and M. Saeki. Supporting requirements change management in goal oriented analysis. In *Proc. of RE'08*, pages 3–12, 2008.
23. H. Ural, R. L. Probert, and Y. Chen. Model based regression test suite generation using dependence analysis. In *Proc. of the 3rd Int. Workshop on Advances in Model-based testing*, pages 54–62, 2007.
24. A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'2004*, pages 148–157, 2004.
25. W. Wu and T. Kelly. Combining bayesian belief networks and the goal structuring notation to support architectural reasoning about safety. In F. Saglietti and N. Oster, editors, *Computer Safety, Reliability, and Security*, volume 4680 of *Lecture Notes in Computer Science*, pages 172–186. Springer Berlin / Heidelberg, 2007.

APPENDIX E

Noname manuscript No.
(will be inserted by the editor)

Dealing with Known Unknowns: A Goal-based Approach for Understanding Complex Systems Evolution ⋆

Fabio MASSACCI, Le Minh Sang TRAN

Università degli Studi di Trento, I-38100 Trento, Italy
e-mail: surname@disi.unitn.it

The date of receipt and acceptance will be inserted by the editor

Abstract Enterprises must cope with evolution to addresses changing business objectives, changing regulations and changing threats. In many cases such changes are not completely unknown: the ongoing discussion of a standard body can show that 2-3 proposals might emerge, albeit might not be clear which one will finally win.

In this paper we tackle the fundamental issue of modeling and reasoning about the future evolution of the business and security goals of an organization in presence of uncertainty of known outcomes.

This work describes a generic approach explicitly representing goal evolutions in terms of controllable and observable rules and in which probability estimates can be validated by a game-theoretic semantics between stakeholders and nature.

In this setting it is important to identify which are the business activities that must be implemented to guarantee the best chances of success (maximum belief) or minimize the risk of wasting money (residual risk). We specialize the set-up for a goal-based language where we provide a graphical language but also efficient algorithm that allows to reason about the unrolling of probabilities without a combinatorial explosion.

In order to illustrate the applicability of the approach we discuss a case study from Air Traffic Management for the deployment of AMAN in air-traffic control procedures.

⋆ It is an extended and revised version of [38]. It is partly supported by the European Commission under projects EU-FET-IP-SECURECHANGE, and EU-IP-NESSOS. We would like to thank F. Paci at the University of Trento, A. Tedeschi, V. Meduri, M. Felici, S. Pozzi at DBL Srl and the other participants to the ATM validation sessions for many useful comments. The numbers presented in this paper are only provided as examples and do not represent in any way the opinion of the experts' organizations.

Key words enterprise modeling, software engineering, observable and controllable rules

1 Introduction

> "...There are known unknowns: that is to say, there are things that we now know we don't know..."
>
> — Donald Rumsfeld, United States Secretary of Defense

The term *software evolution* has been introduced by Lehman in his work on laws of software evolution [17, 18], and was widely adopted since 90s. Recent studies in software evolutions attempt to understand causes, processes, and effects of the phenomenon [2, 14, 16]; or focus on the methods, tools that manage the effects of evolution [19, 29, 36].

In the domain of software systems [12, 15, 27, 33, 42], evolution refers to a process of continually updating software systems in accordance to changes in their working environments such as business requirements, regulations and standards. While some evolutions are unpredictable, many others can be predicted albeit with some uncertainty (e.g. a new standard does not appear overnight, but is the result of a long process).

It is now widely accepted that in order to fully understand an enterprise system we can no longer consider simply its IT structure. We face a socio-technical system [31] "that involve complex interactions between software components, devices and social components (people or groups of people), not as users of the software but as players engaged in common tasks" [11].

This is particularly true for large systems of systems such as the Air Traffic Management "system" (ATM for short). Modelling the key objectives on an Air Traffic Control Organization requires to include both human and system actors and, before digging into detailed software features, requires the ability to reason about high-level strategic assignments of goals to those human and system actors.

In the ATM setting changes are often organizational changes that involves complex subsystems as a whole. For example, the SESAR Open Sky initiative foresee the introduction of an Arrival Manager (a system) in order to replace some of the activities by the Sequence Manager (a human). Still this system relies on decisions by other humans. Evolution is therefore not represented in terms of software features but rather in assigning high-level mission critical goals such as "maintain aircraft separation" to different actors.

The potential evolutions of such large and complex system is not completely unpredictable, as it often involves significant multi-party (or even multi-state) negotiations. Stakeholders with experience and high-level positions have a good visibility of the likely alternatives, the possible but unlikely solutions, and the politically impossible paths. For example, the Federal Aviation Authority (FAA) document of the System Wide Information Management (SWIM) for Air Traffic Management (ATM) lists a number of potential technical alternatives that depends from high-level decisions (*e.g.,* the existence of an organizational agreement for nationwide identity management of SWIM users).

Therefore, it is possible to model the evolution of mission-critical requirements at enterprise level when such evolution is known to be possible, but it is unknown whether it would happen: the *known unknown*. We target at capturing what Loucopoulous and Kavakli [21] identified as the knowledge shared by multiple stakeholders about *"where the enterprise is currently"*, *"where the enterprise wished to be in the future"*, and *"alternative designs"* for the desired future state.

Unfortunately, an ATM organization cannot wait that the unknowns becomes known. The process of tendering and organizational restructuring requires a significant amount of time and planning. Therefore decision makers at high-level must essentially bet on the final organizational solution and possibly minimize the risks that the solutions turns out to be wrong.

In this respect it is important to provide a sound quantitative analysis which was identified by Dalal et al. in [8] as one of the current weaknesses of enterprise modelling systems.

1.1 The Contribution of This Paper

Our ultimate goal is to support the decision maker in answering such a question "Given these anticipated evolutions, what is a the solution to implement an evolution-resilient system?". By a solution we mean here either a software or an organizational solution such as assigning a task to an actor instead of another.

To address this objective we set up a global framework for modeling and reasoning on evolution in socio-technical systems as follows:

- we introduce the idea of modelling evolutions in terms of two kinds of evolution rules: *controllable* and *observable* rules that are applicable to arbitrary enterprise and high-level requirements models (from problem frames to goal models);
- we identify a game-theoretic based explanation for probabilities of an observable evolution in terms of a game between reality (that finally decides what is happening), stakeholders (who provide likelihood information on outcomes) and designers (who bet against the odds);
- we provide two quantitative metrics to help the designer in deciding optimal things to implement for the system-to-be;
- for the particular case of goal models we provide an optimal algorithm for calculating such metrics in case of complex scenarios with multiple alternatives;
- also briefly describe a graphical notation for representing such evolution rules that has been validated in a number of sessions with ATM experts.

The rest of this paper is organized as follows. In the subsequent section (§2) we describe a case study in the field of Air Traffic Management, by which examples are extracted to illustrate our approach. Next, we discuss the basic idea of our generic approach (§3) to deal with known unknown. Then we instantiate it to a concrete syntax based on goal models that we have used for validation with ATM experts (§4). We additionally describe a game-theoretic semantics to account for the evolution probability (§5). The paper is then continue by a discussion of two

quantitative metrics, max belief and residual risk (§6), to support decision makers in selecting optimal configuration for the enterprise model in practical scenarios of evolution (§7). For our concrete instantiation with goal models we also present an incremental algorithm to calculate the two proposed metrics (§8). Finally, we discuss related works (§9) and conclude the paper (§10).

2 Air Traffic Management Case Study

In order to make the discussion more concrete we present here a case study on the Air Traffic Management (ATM) system that we have carried out in the framework of the EU project SECURECHANGE [28]. The focus in this paper is the deployment of the Arrival Manager (AMAN) into an air traffic control room. AMAN, DMAN (Departure Manager), and SMAN (Surface Manager) are queue management tools, introduced by the SESAR Open Sky initiative. These support tools will substitute some human activities to deal with increasingly traffic loads at terminals, while still guaranteeing better performance, efficiency, and safety.

As the name suggests, the AMAN is a ground-based tool suggesting to the Air Traffic Controllers (ATCO) an optimal arrival sequence of aircrafts and providing support in estimating the optimal aircraft approach route. The critical mission of AMAN is to support ATCO to manage the arrival traffic safely and efficiency, *i.e.* by maintaining an appropriate separation between aircrafts. To achieve this high level objective, several processes and activities are performed. Here we only focus on two of them, *applying an optimal arrival sequence*, and *collaborating with other units*.

AMAN calculates an optimum arrival sequence considering many constraints such as flight profiles, aircraft types, airspace and runway condition, inbound flow rate, as well as meteorological conditions. The final sequence is approved by ATCOs. Then, the AMAN generates various kinds of advisories for ATCO to send to pilots *e.g.,* time to lose or gain, top-of-descent, track extension, while their execution is continuously monitored to react to eventual violations. The sequence and other relevant data are exchanged with adjacent sectors to improve collaboration and reduce conflicts.

Fig. 1 summarizes these objectives in a goal model. We will define it formally in later section but the intuitive understanding is that in order to achieve the goal depicted in a node we need to achieve the goals described in its children. In the ATM domain they are sometimes called influence diagrams.

The ATM operational environment, however, is continuously evolving due to numerous causes such as the rapidly increasing of traffic load, the new performance and safety standards, as well as the need of tighter collaboration between ATM systems. To efficiently cope with evolutions, potential changes must be foreseen, for instance:

- AMAN should support what-if-probing and online simulation.
- AMAN should support trajectory prediction, *i.e.* Expected Time of Arrival
- AMAN should support advanced advisories generation: heading, speed instructions.

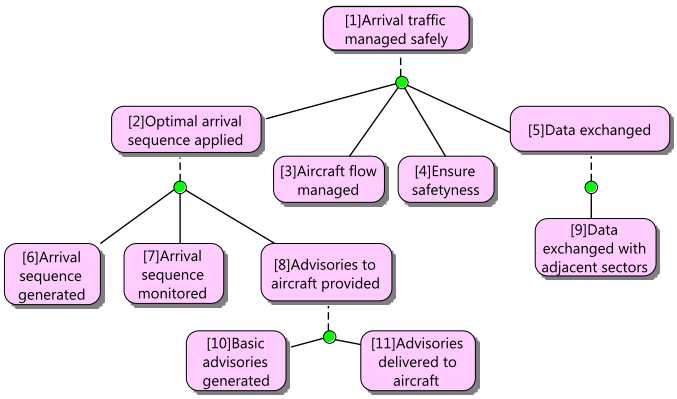


Fig. 1 Goal model of the AMAN case study without potential evolutions.

- AMAN should be interoperable and optimized at European level, by supporting SESAR 4D operations such as Controlled Time of Arrival (CTA) and Required Time of Arrival (RTA) negotiation.
- AMAN should be able to exchange with DMAN, SMAN in a same and other airports.
- AMAN should integrate with other monitoring and conflict detection tools such as Monitoring Aids (MONA), Medium Term Conflict Detection (MTCD).

These objectives are not compulsory at current, but they might be required in future. A cheap implementation of AMAN ignoring all potential evolutions might cause a gigantic cost to update deployed systems. However, implementing all of them can be a waste of money as we are not sure they are really necessary.

In subsequent sections we describe our approach to deal with these known unknown in fulfilling the overall ATM enterprise objectives.

3 Controllable and Observable Evolutions

To improve the readability of the paper, Table 1 summarizes important terms used throughout this work, which can be formally or informally defined later in the text.

Enterprise goals, mostly in textual format, express the enterprise mission-critical objectives. The achievement of each of objectives is a requirement that all future solutions must meet. They can be captured by specific languages/toolsets of enterprise modeling such as IDEF [32], IEM [22], GRAI/GIM [9], ARIS/EPC [34], UEML [39], software oriented languages such as UML or more research-oriented languages such as KAOS or i* [41]. These models comprise various kinds of entities depended on the modeling languages and relations among these entities. For instance: KAOS uses ‘goal’ to represent requirements, whereas, UML uses ‘use-case’ for similar purposes.

Toward to a solution for the evolution of enterprise objectives without sticking into any specific languages, we define an enterprise model at the meta-level point of view as follows:

Table 1 Notions and concepts

Term	Notion	Definition
Enterprise model	EM	is a set of entities and relations to model enterprise objectives (or system requirements).
Subpart	SM	is a sub set of an enterprise model.
	SM^{α}	is the original model of the α^{th} observable rule.
	SM_i^{α}	is the ith evolution possibility of the α^{th} observable rule.
	$SM_{i,j}^{\alpha}$	is the jth design alternative of SM_i^{α}.
Enterprise model evolution (evolution)		is a set of changes in the enterprise model, including addition, remove of objectives. Objective modification is treated as a series of remove and delete.
Evolution rules	r_o, r_c	is a set of evolution possibilities.
	$r_{o\alpha}$	is the α^{th} observable rule.
	$r_{c\alpha i}$	is the controllable rule applied to the ith possibility of the observable rule $r_{o\alpha}$
Evolution possibility (possibility)		is a potential evolution. In a same evolution rule, only one possibility is able to happen.
Evolution probability	p	is the belief of stakeholder about the likelihood by which an evolution possibility will happens.
Design alternative (alternative)	*DA*	is a set of elements of an enterprise model that are necessary to fulfil all objectives.
Design alternative set	$SDA(\mathcal{C})$	is a set of design alternatives that are supported by a configuration $\mathcal{C}$.
Final choice (configuration)	$\mathcal{C}$	is a set of elements in the enterprise model that are chosen to implement an enterprise system. A configuration can comprise one or more design alternatives.

Smaller words in parentheses are short names (or aliases) of the terms in normal text.

Definition 1 (Enterprise model) *An enterprise model* EM *is a tuple* $\langle E, R\rangle$ *where* E *is set of entities, and* R *is set of relations among them.*

Evolution of enterprise objectives refers to changes of objectives of an enterprise, basically falling into two categories, *controllable evolution* and *observable evolution*. This justification is based on the factor (both human and non-human factor) that initiate the changes.

- *Controllable evolutions* are subject to system designers who intentionally change the system design in order to fulfil high level enterprise objectives. In other words, they are designers' moves to identify design alternative to implement an enterprise system. Later, the most "optimal" alternative is chosen to the next development phases. Such decision are based on tool-support analyses (both qualitative and quantitative) and expertise experiences.
- *Observable evolution*, whereas, are out of control of "inside people" *i.e.* who directly involve in the development process such as domain experts, designers and so on. However, these evolutions can be forecasted with a certain level of confidence. As the impact of the observable evolutions, new objectives are in-

troduced, or objectives' status is changed *e.g.,* from 'optional' to 'compulsory', or from 'compulsory' to 'obsoleted'.

In an evolution-aware developing process, the selection of optimal design alternative(s) for the implementation of an enterprise system should carefully take into account observable evolutions, otherwise it may be incapable to support the enterprise objectives in future.

To support decision making in dealing with the evolution of enterprise objectives, these two kinds of evolutions are incorporated into enterprise models in terms of evolution rules. The basic idea of evolution rules is to capture the snapshots of enterprise objectives before and after evolutions. Evolutions rules are formally defined as follows:

Definition 2 (Controllable rule) *A controllable rule r_c is a set of tuples $\langle EM, EM_i \rangle$ that consists of an original model EM and its possible design alternative EM_i.*

$$r_c = \bigcup_i \left\{ EM \xrightarrow{*} EM_i \right\}$$

Example 1 (Controllable rule) Consider again Fig. 1, the objective g_5-Data exchanged is refined into only one lower level objective g_9-Data exchanged with adjacent sectors. However an organization might chose to integrate the DMAN in the process and thus might choose between two other goals $g_{12}-$Basic data exchanged with DMAN and $g_{13}-$Advanced data exchanged with DMAN.

Definition 3 (Observable rule) *An observable rule r_o is a set of triples $\langle EM, p_i, EM_i \rangle$ that consists of an original model EM and its potential evolution EM_i. The probability that EM evolves to EM_i is p_i. All these probabilities should sum up to one.*

$$r_o = \bigcup_i \left\{ EM \xrightarrow{p_i} EM_i \right\}$$

Example 2 (Observable rule) Consider again Fig. 1, the objective g_5-Data exchanged is refined into only one lower level objective g_9-Data exchanged with adjacent sectors. One potential evolution for is that a regulatory body requires that the organization should also achieve both goal $g_{12}-$Basic data exchanged with DMAN and goal $g_{13}-$Advanced data exchanged with DMAN albeit for different functionalities. If this possibility actually happens the organization should meet g_9, g_{12}, g_{13} in order to meet g_5. However, the discussion in the regulatory body is still quite open and another option might be to actually leave partners to chose whether to impose g_9-Data exchanged with adjacent sectors and $g_{12}-$Basic data exchanged with DMAN and leave operators the choice of implementing g_9 and $g_{13}-$Advanced data exchanged with DMAN. This latter option, which combines the one from Example2, is denoted by EM_1 as it is the most likely outcome, the former we denote it by EM_2. Obviously, there is also another possibility: EM might not evolve and therefore remain unchanged (decision of regulatory body is post-poned after first deployment trials of DMAN). These possibilities exclusively happen with some uncertainties. Thus, the observable rule is would be the following one.

$$r_{o1} = \left\{ EM \xrightarrow{p_1} EM_1, EM \xrightarrow{p_2} EM_2, EM \xrightarrow{1-p_1-p_2} EM \right\}$$

■

Example 3 (Controllable rule) In previous example, EM_1 has two possible design alternatives as g_5 is either refined into $\{g_9, g_{12}\}$, or $\{g_9, g_{13}\}$. Let's name these goal sets $EM_{1,1}$ and $EM_{1,2}$, respectively. Then, the controllable rule is as follows.

$$r_{c1} = \left\{ EM_1 \xrightarrow{*} EM_{1,1}, EM_1 \xrightarrow{*} EM_{1,2} \right\}$$

■

Models incorporating evolution rules are called *evolutionary models*.

4 Specialization of the Approach on Goal-Based Model

In order to be concretely used by the stakeholders, the proposed approach for a generic enterprise model with evolution rules needs to be instantiated to a concrete syntax. Our choice is to use goal models. The first reason is obviously the expertise that we could find at our own university. The second, possibly more important, is that models for representing goals and objectives are well known in the ATM domain (possibly under the name of "influence diagrams").

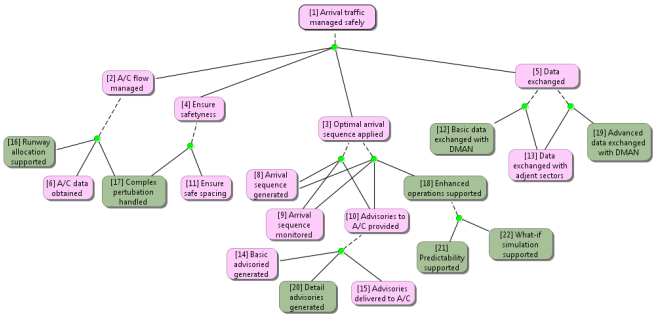
In order to make our syntax concrete, some notions and concepts discussed in generic sense are rephrased in the context of goal-based language. We further discuss the visual presentation of evolution rules into goal models.

Basically, the general idea of goal-based approaches is the employment of goal notion to study enterprise objectives or goals. Goals are refined (or decomposed) into many subgoals in the sense that parent goal can be achieved if either all subgoals are fulfilled (*AND-decomposition*). In this manner, goals are recursively decomposed until operational goals (or tasks) that can be done by either software systems or humans. Hence a goal-based enterprise model can be formally written as follows.

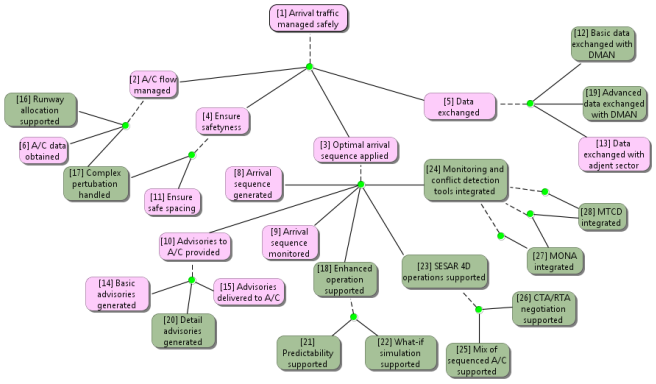
Definition 4 (Goal-based enterprise model) *A goal-based enterprise model is a tuple* $\langle G, De \rangle$ *in which* G *is a set of goals, and* $De \subset G \times 2^G$ *is a set of AND-decomposition relations between goals.*

Traditional goal models also include the notion of (*OR-decomposition*) where the same goal can be fulfilled in different ways. We will not use in this setting or-decompositions as they are better characterized by design alternatives in terms of controllable rules (and better understood in this form by experts). A goal model itself contains several design alternatives. Each design alternative is combination of all different ways an arbitrary goal is decomposed. Since a goal is achieved when all its refined goals is fulfilled, a final choice for the various design alternatives can also be represented by the set of leaf goals required to fulfil all top goals.

The evolutionary goal model is intuitively a specification of the evolutionary enterprise model aforemention, where the controllable rules are implicitly represented by the different ways in which goals can be decomposed. The *evolution-dependent* relations between observable rules are also implied based on the decomposition. If rule r_{o1}, r_{o2} respectively apply to goal g_1, g_2, and g_2 is a (direct or in direct) child of g_1, then r_{o2} is *evolution-dependent* to r_{o1}. We additionally define



(a) AMAN Evolution 1



(b) AMAN Evolution 2

The right hand corner of the two models corresponds to the different evolutions of the model that we have described in Example 2.

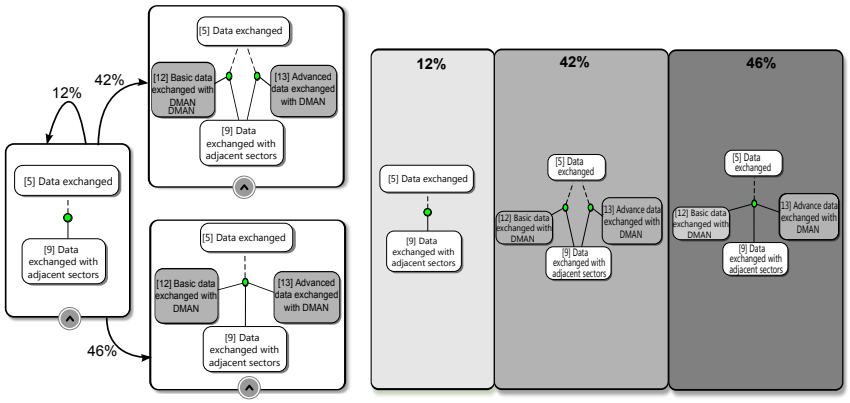
Fig. 2 Alternative Evolutions for AMAN Deployment.

a *subpart* of a goal model is an arbitrary goal and all its directly and indirectly children. The evolutionary goal model, therefore, is a tuple of an original goal model and a set of observable rules, $\langle EM, \mathcal{R}_o \rangle$.

It is not feasible to represent all evolutions in the same diagram. In order to understand the level of complexity we show in Fig. 2 two alternative models in which a number of observable evolutions rules have been triggered. In comparison with Fig. 1 the new goals have been marked in green. It is clearly impossible to identify the two simple evolution rules that we have described. These evolutions must be represented as local evolutions of the corresponding fragment of the models.

Here we outline how evolution rules are visualized in an evolutionary goal model, more discussion can be found in a technical report [37]. Fig. 3 illustrates two graphical representations of an observable rule, *tree-like* and *swimlane-based*, taken from Example 2.

Fig. 3(b), meanwhile, describes the swimlane-based representation where original model is on the left side, and all the possible evolutions are located next to the right. Evolution probability is indicated by the label and the shade of swimlanes.



(a) Tree-like representation (b) Swimlane-based representation

Fig. 3 Visualization of evolution rules.

The tree-like representation, Fig. 3(a), basically is a tree-like directed graph, where a node represent a subpart in the goal model. The directional connection between two nodes means the source node evolves to target node. In other words, the source node is the original model (or subpart), and target node is the evolved model (subpart). A connection is labeled with an evolution probability p_i. In two validation sessions with ATM experts the tree-like model was found more intuitive and we have therefore decided to implement it in our modelling CASE tool.

5 Game-Theoretic Approach Accounting for Evolution Probability

Our notion of observable rules includes an associated probability. Probabilities are often taken for granted by engineer and scientists but in our setting it is important to understand the exact semantics of this notion.

Basically, there are two broad categories of probability interpretation, called "physical" and "evidential" probabilities. Physical probabilities, in which frequentist is a representative, are associated with a random process. Evidential probability, also called Bayesian probability (or subjectivist probability), are considered to be degrees of belief, defined in terms of disposition to gamble at certain odds; no random process is involved in this interpretation.

To account for probability associated with an observable rule, we can use the Bayesian probability as an alternative to the frequentist because we have no event to be repeated, no random variable to be sampled, no issue about measurability (the system that designers are going to build is often unique in some respects). However, we need a method to calculate the value of probability as well as to explain the semantic of the number. Since probability is acquired from the requirements/objectives eliciting process involving the Domain Experts, we propose using the game-theoretic method in which we treat probability as a price. It seems to be easier for Domain Experts to reason on price (or cost) rather than probability.

The game-theoretic approach, discussed by Shafer et al. [35] in Computational Finance, begins with a game of three players, *i.e.* Forecaster, Skeptic, and Real-

ity. Forecaster offers prices for tickets (uncertain payoffs), and Skeptic decides a number of tickets to buy (even a fractional or negative number). Reality then announces real prices for tickets. In this sense, probability of an event E is the initial stake needed to get 1 if E happens, 0 if E does not happen. In other words, the mathematics of probability is done by finding betting strategies.

Since we do not deal with stock market but with the design of evolving enterprise goals, we need to adapt the rules of the game. Our proposed game has three players: *Domain Expert*, *Designer*, and *Reality*. For the sake of brevity we will use "he" for Domain Expert, "she" for Designer and "it" for Reality. The sketch of this game is denoted in protocol 1.

Protocol 1
Game has n round, each round plays on a software C_i
FOR i = 1 to n
Domain Expert announces p_i
Designer announces her decision d_i: believe, don't believe
If Designer believes
$K_i = K_{i-1} + M_i \times (r_i - p_i)$
Designer does not believe
$K_i = K_{i-1} + M_i \times (p_i - r_i)$
Reality announces r_i

The game is about the desire of Domain Expert to have a software *C*. He asks Designer to implement *C*, which has a cost of *M*\$. However, she does not have enough money to do this. So she has to borrow money from either Domain Expert or a bank with the return of interest (ROI) *p* or *r*, respectively.

Domain Expert starts the game by announcing *p* which is his belief about the minimum ROI for investing *M*\$ on *C*. In other words, he claims that *r* would be greater than *p*. If *M* equals 1, *p* is the minimum amount of money one can receive for 1\$ of investment. Domain Expertshows his belief on p by a commitment that he is willing to buy C for price $(1+p)M$ if Designer does not believe him and borrow money from someone else.

If Designer believes Domain Expert, she will borrow *M* from Domain Expert. Later on, she can sell *C* to him for $M(1+r)$ and return $M(1+p)$ to him. So, the final amount of money Designer can earn from playing the game is $M(r-p)$.

If Designer does not believe Domain Expert, she will borrow money from a Bank, and has to return $M(1+r)$. Then, Domain Expert is willing to buy *C* with $M(1+p)$. In this case, Designer can earn $M(p-r)$.

Suppose that Designer has an initial capital of K_0. After round i-*th* of the game, she can accumulate either $K_i = K_{i-1} + M(r-p)$ or $K_i = K_{i-1} + M(p-r)$, depend on whether she believes Domain Expert or not. Designer has a winning strategy if she can select the values under her control (the *M*\$) so that her capital never decrease, intuitively, $K_i >= K_{i-1}$ for all rounds.

The law of large numbers here corresponds to say that if unlikely events happen then Designer has a strategy to multiply her capital by a large amount. In other words, if Domain Expert estimates Reality correctly then Designer has a strategy for avoid overshooting her budget.

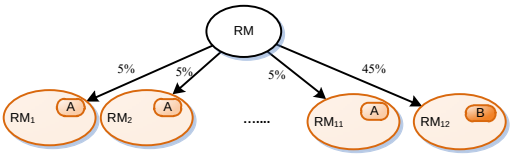


Fig. 4 The long-tail problem.

6 Quantitative Metrics: Max Belief and Residual Risk

As consequence of actual occurences of evolutions possibilities, a final design choice may or may not be able to fulfil the enterprise's objectives. In order to select among the various alternatives we need a quantitative metric that can guide the decision maker.

The first intuitive measure is the total probability that a final configuration survived, called *Total Belief*: the sum of probabilities of all possibilities where the final choice fulfill the enterprise's objectives. If complexity is not a problem it is also easy to compute: just unroll all probabilities combining them into one big observable rule. Then gather every possible model that (i) is resulting from the evolution, (ii) includes the final design choice and (iii) fulfils the root objectives of the enterprise.

Unfortunately, this measure might lead to a severe *long-tail* problem. This problem, firstly coined by Anderson [1], is present when a larger than normal population rests within the tail of the distribution. A long-tail example is depicted in Fig. 4 where an enterprise model EMmight evolve to several potential possibilities with very low probabilities (say, eleven possibilities with 5% each), and another extra possibility with dominating probability (say, the twelfth one with 45%).

Suppose that an element A appears in the first eleven possibilities, and an element B appears in the last twelfth possibility. Apparently, A is better than B due to A's total belief is 55% which is greater than that of B, say 45%. Arguing that A is better than B or versa is still highly debatable. Ones might put their support on the long tail [1], and ones do the other way round [10].

Our game semantics allows us to make an informed choice: at the end of the day, only one possibility becomes effective (*i.e.* chosen by Reality). If we thus consider every single possibility to be chosen, the twelfth one (45%) is the one with the highest pay-off by the Domain Expert. The other possibilities offers a substantial less chance of making money: Domain Expertwill only pay a 5% ROI for any of the other possibilities. It is true that globally A might be chosen by any of the 11 alternatives, but reality will not realized them all together. It will only chose one and each of them only have a ROI of 5%. Choosing the largest ROI is one of the possible alternatives.

However, we need to be careful since Designer must invest M to build B before knowing whether it would pay off. So what is interesting for the Designer is a measure of the risks that its choice might turn out to be sour. We are currently investigating a number of alternatives. A possible solution is to consider the dual of the MaxBelief and namely the MaxRisk as the highest chances of a possibility

where the configuration is not useful. To be more conservative, in this paper we consider as risk measure the complement of the Total Belief and namely the sum of the total chances that a configuration would turn out to be utterly useless.

Building on the above considerations, we introduce two quantitative metrics: *Residual Risk*[1] and *Max Belief* as follows.

Max Belief (MaxB): of an configuration $\mathcal{C}$ is a function that measures the maximum belief supported by Domain Expert such that $\mathcal{C}$ is useful after evolution happens.

Residual Risk (RRisk): of an configuration $\mathcal{C}$ is the complement of total belief supported by Domain Expert such that $\mathcal{C}$ is useful after evolution happens. In other words, residual risk of $\mathcal{C}$ is the total belief that $\mathcal{C}$ is not useful when evolutions happen.

They offer two independent dimensions upon which a designer can chose.

Given an evolutionary enterprise model $\langle EM, r_o, r_c \rangle$, max belief and residual risk can be formally defined as follows.

$$\begin{aligned} MaxB(\mathcal{C}) &\triangleq \max_{\forall EM \xrightarrow{p_i} EM_i . EM_i \in SDA(\mathcal{C})} p_i \\ RRisk(\mathcal{C}) &\triangleq 1 - \sum_{\forall EM \xrightarrow{p_i} EM_i . EM_i \in SDA(\mathcal{C})} p_i \end{aligned} \tag{1}$$

where $SDA(\mathcal{C})$ is the set of design alternatives which a configuration $\mathcal{C}$ comprises (or support), also called as *design alternative set* of a configuration $\mathcal{C}$.

The residual risk, as discussed, is the complement of total belief. Hence, for convenience, the *Total Belief* of $\mathcal{C}$ is denoted as:

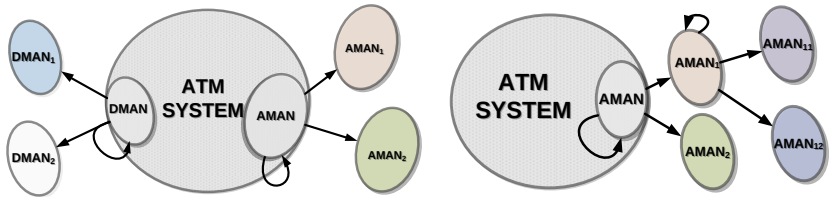
$$\overline{RRisk(\mathcal{C})} = 1 - RRisk(\mathcal{C})$$

The selection between two configurations based on max belief and residual risk is obvious: "higher max belief, and lower residual risk". However, it is not always a case that a higher max belief configuration has lower residual risk. Thus decision makers should understand which criterion is more important. In this sense, these metrics could be combined using weighted harmonic mean. Suppose that w_1, w_2 are weights of max belief and residual risk, respectively. The harmonic mean is defined as follow.

$$H(\mathcal{C}) = \frac{w_1 + w_2}{\frac{w_1}{MaxB(\mathcal{C})} + \frac{w_2}{1 - RRisk(\mathcal{C})}} = (1+\beta)\frac{MaxB(\mathcal{C}) \cdot (1 - RRisk(\mathcal{C}))}{\beta \cdot MaxB(\mathcal{C}) - RRisk(\mathcal{C}) + 1}$$

where $\beta = {}^{w_1}/_{w_2}$ means max belief is β times as much important as residual risk.

[1] One should not confuse this notion of residual risk with the one in security risk analysis studies which is different in nature.



(a) Multi-part evolution scenario (b) Multi-step evolution scenario

The left (a) exemplifies a multi-part evolution, where a big ATM system has two separated subsystems (parts), AMAN and DMAN, which may evolve independently. Meanwhile, the right (b) illustrates a multi-step scenario. After AMAN evolves to AMAN_1, it continues to evolve to either AMAN_{11} or AMAN_{12}.

Fig. 5 Complex evolution scenarios.

7 More Complex Evolution Scenarios

In practice, evolutions can fall into one of two scenarios: *multi-part* and *multi-step* (or combination of these scenarios).

The *multi-part scenario* indicates evolutions in different parts of a big enterprise model, as illustrated in Fig. 5(a). This is a case that a system comprises of several subsystems which are relatively independent. For instance, in the whole ATM system, the AMAN subsystem (Arrival Management System) is more or less independent with the DMAN (Departure Management System) subsystem even though they can exchange data. Thus AMAN can evolves independently with DMAN. We call evolution in subparts of the model as *local evolution*.

The *multi-step evolution* scenario determines the case that the system is iteratively evolving. In Fig. 5(b), the AMAN subsystem of ATM, after evolving to AMAN_1, may continue to evolve to either AMAN_{11} or AMAN_{12}. Suppose that the evolution of AMAN to AMAN_1 or AMAN_2 is subject to an observable rule r_{o1}; and the evolution of AMAN_1 to AMAN_{11} or AMAN_{12} is subject to an observable rule r_{o2}. The global evolution of AMAN, in this case, is a *2-step evolution*. Obviously, r_{o2} can only be effective only if r_{o1} happens in the first step/phase, AMAN $\longrightarrow$ AMAN_1. We call the relationship between r_{o1} and r_{o2} *evolution-dependent* relationship, which are formally defined as follows.

Definition 5 (Evolution-dependent relation) *Given two observable rules* $r_{o1} = \bigcup \left\{ EM \xrightarrow{p_i} EM_i \right\}$ *and* $r_{o2} = \bigcup \left\{ EM' \xrightarrow{p_j} EM'_j \right\}$*, the rule* r_{o2} *is evolution-dependent to the rule* r_{o1}*, denoted as* $r_{o1} \overset{i}{\rightsquigarrow} r_{o2}$ *if there exists a possibility* $EM \xrightarrow{p_i} EM_i$ *of* r_{o1} *such that* EM' *is a subset (or equal) of* EM_i*.*

$$r_{o1} \overset{i}{\rightsquigarrow} r_{o2} \Leftrightarrow \exists EM \xrightarrow{p_i} EM_i \in r_{o1}.EM' \subseteq EM_i \tag{2}$$

To this end we define the evolutionary enterprise model that take into account all complex evolution scenarios as follows.

Definition 6 (Evolutionary enterprise model) *An evolutionary enterprise model* eEM *is a quadruplet* $\langle EM, \mathcal{R}_o, \mathcal{R}_c, Dep \rangle$ *where:*

- EM *is the original enterprise model,*
- $\mathcal{R}_o$ *is set of observable evolution rules.*
- $\mathcal{R}_c$ *is a set of controllable rules applying to the original enterprise model and other evolved ones.*
- $Dep \subseteq \mathcal{R}_o \times \mathcal{R}_o \times \mathbb{N}$ *is a set of evolution-dependent relations between observable rules.*

$$\begin{aligned} \mathcal{R}_o &= \bigcup \left\{ r_{o\alpha} = \bigcup \left\{ SM^{\alpha} \xrightarrow{p_i^{\alpha}} SM_i^{\alpha} \right\} \right\} \\ \mathcal{R}_c &= \bigcup \left\{ r_{c\alpha i} = \bigcup \left\{ SM_i^{\alpha} \xrightarrow{*} SM_{ij}^{\alpha} \right\} \right\} \\ Dep &= \bigcup \left\{ r_{o\alpha} \overset{i}{\rightsquigarrow} r_{o\beta} \right\} \end{aligned} \tag{3}$$

where $SM^{\alpha} \subseteq EM \vee \exists SM^{\alpha'} \xrightarrow{p_i^{\alpha'}} SM_i^{\alpha'} . SM^{\alpha} \subseteq SM_i^{\alpha'}$

The calculation of the max belief and residual risk for a given configuration $\mathcal{C}$ on a given evolutionary enterprise model eEM is complicated by the need to consider both multi-part and multi-step evolutions.

However, once we have defined a semantics for the basic set-up of probabilities, we can now leverage on the mathematics behind the theory or probabilities.Thus we can use the mechanisms of conditional probabilities for multi-step evolution and of independent combination of events for multi-part evolution.

The max belief and residual risk of a given configuration $\mathcal{C}$ and an evolutionary enterprise model $eEM\langle EM, \mathcal{R}_o, \mathcal{R}_c, Dep\rangle$ are defined as follows.

$$\begin{aligned} MaxB(\mathcal{C}) &\triangleq \max_{\forall(\bigwedge_\alpha SM^\alpha \xrightarrow{p_i^\alpha} SM_i^\alpha).\, \bigcup_\alpha SM_i^\alpha \in SDA(\mathcal{C})} Pr\left(\bigwedge_\alpha SM^{\alpha} \xrightarrow{p_i^{\alpha}} SM_i^{\alpha} \right) \\ RRisk(\mathcal{C}) &\triangleq 1 - \sum_{\forall(\bigwedge_\alpha SM^\alpha \xrightarrow{p_i^\alpha} SM_i^\alpha).\, \bigcup_\alpha SM_i^\alpha \in SDA(\mathcal{C})} Pr\left(\bigwedge_\alpha SM^{\alpha} \xrightarrow{p_i^{\alpha}} SM_i^{\alpha} \right) \end{aligned} \tag{4}$$

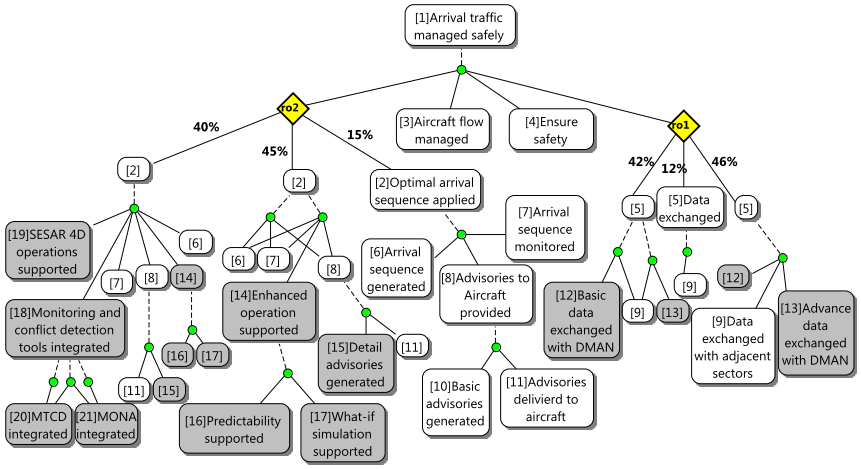
8 An Incremental Algorithm to Calculate Max Belief and Residual Risk

Developing an algorithm to calculate max belief and residual risk, as in Formula 4, for an evolutionary goal model is not practical. With the heuristic assumption that observable rules in different parts are independent we can efficiently eliminate recursion by transforming the model into a suitable hypegraph and use efficient hyperpath algorithms.

An evolutionary goal model is converted in to an *evolutionary hypergraph* as follows.

Definition 7 (Evolutionary hypergraph) *The hypergraph* $\mathcal{G}\langle V, E\rangle$ *of an evolutionary goal model* $\langle EM, \mathcal{R}_o\rangle$ *is constructed as follows:*

- *For each goal* g*, create a goal node* g.



Round rectangles represent goal nodes, circles are compound nodes, and diamonds denotes observable nodes. Goals with a same number refer to the same objective.

Fig. 6 The hypergraph of the case study described in §2 with evolution rules embedded.

- *For each decomposition* $\langle g, S_g \rangle$*, create a compound node* c*. Then create a dotted edge* $\langle c, g, 1 \rangle$ *connecting* c *to* g*. For each goal* $g' \in S_g$*, create a full edge* $\langle g', c, 1 \rangle$ *connecting* g' *to* c*.*
- *For each observable rule* $\bigcup \left\{ SM^{\alpha} \xrightarrow{p_i^{\alpha}} SM_i^{\alpha} \right\}$*, create an observable node* o*. Then create full edges* $\langle o, g_{\alpha}, p_i^{\alpha} \rangle$ *where* g_{α} *denotes the top goal of* SM^{α}*.*

This hypergraph has three kinds of nodes: goal node, observable node, and compound node. These kinds of nodes orderly correspond to goals, observable rules, and decomposition of goals. Besides, there are two kinds of edges: full edges and dotted edges. The full edge connects an observable node or compound node to another observable or goal node. The full edge connecting an observable node to a goal node denotes a potential evolution possibility of the goal node. Therefore, goal nodes connected to an observable node always have the same name, determining several potential evolution possibilities of the observable rule corresponding to this observable node. Meanwhile, full edges connecting a compound node to other nodes together with a dotted edge connecting this compound node to another goal node represent a decomposition.

Example 4 (Hypergraph) To the illustrative purpose, Fig. 6 describes the hypergraph of the case study with two observable rules r_{o1} and r_{o2} for goal g_2-Optimal arrival sequence applied and goal g_5-Data exchanged, respectively. Notice that, goals with a same number refer to the same objective, and only one of them is fully labeled to save the space.

In this figure, white goals indicate objectives identified at current time. Meanwhile, gray goals denote objectives introduced if evolution happens. In rule r_{o1}, discussed in Example 2, the original subpart $\{g_5 \leftarrow g_9\}$ might evolve to either

$\{g_5 \leftarrow \{g_9, g_{12}\}, g_5 \leftarrow \{g_9, g_{13}\}\}$ or $\{g_5 \leftarrow \{g_9, g_{12}, g_{13}\}\}$ with a probability of 46% and 42%, respectively. The original part might stay unchanged with a probability of 12%. Similarly, in r_{o2} the original subpart $\{g_2 \leftarrow \{\ldots\}\}$ might also evolves to two other possibilities with probabilities of 45% and 40%. The remain unchanged probability of r_{o2} is 15%. ∎

Once an evolutionary goal model was transformed into a corresponding evolutionary hypergraph $\mathcal{G}\langle V, E\rangle$, every node x in $\mathcal{G}$ is tagged with a data structure called *design alternative table* (DAT), denoted as $DAT(x)$, holding information for further calculating the max belief and residual risk. $DAT(x)$ is a set of tuples $\bigcup\{\langle S_i, mb_i, rr_i, T_i\rangle\}$, where:

- $S_i \subseteq G$ is a set of leaf goals necessary to fulfil this node.
- mb_i, rr_i are the values of max belief and residual risk, respectively.
- T_i is a set of strings $\langle r_{o\alpha}, i\rangle$ which is the ith possibility of a rule $r_{o\alpha}$. This is used to keep track of the path of evolutions.

Obviously, two tuples in a DAT which have a same T_i are two design alternatives of an observable evolution possibility. Initially, DAT of each node is initialized as follows.

$$DAT(x) \leftarrow \begin{cases} \{\langle\{x\}, 1, 0, \emptyset\rangle\} & \text{if } x \text{ is a leaf goal,} \\ \emptyset & \text{otherwise.} \end{cases} \tag{5}$$

The DATs of leaf nodes are then propagated upward to predecessor nodes (ancestors). This propagation is done by two operators `join`($\otimes$) and `concat`($\oplus$). Also via these operations, DAT of an predecessor node is generated using their successor's DATs.

$$\begin{aligned} &DAT(x_1) \oplus DAT(x_2) \leftarrow DAT(x_1) \cup DAT(x_2) \\ &DAT(x_1) \otimes DAT(x_2) \leftarrow \bigcup_{i,j} \{\, \langle S_i \cup S_j, mb_i \cdot mb_j, 1 - \overline{rr_i} \cdot \overline{rr_j}, T_i \cup T_j\rangle \mid \\ &\qquad \langle S_i, mb_i, rr_i, T_i\rangle \in DAT(x_1), \langle S_j, mb_j, rr_j, T_j\rangle \in DAT(x_2)\} \end{aligned} \tag{6}$$

Lemma 1 *The operator* `join` *and* `concat` *are commutative and associative.*

Proof (Lemma 1) By definition,

$$\begin{aligned} DAT(x_1) \oplus DAT(x_2) &\leftarrow DAT(x_1) \cup DAT(x_2) \\ &= DAT(x_2) \cup DAT(x_1) \rightarrow DAT(x_2) \oplus DAT(x_1) \end{aligned} \tag{7}$$

$$\begin{aligned} (DAT(x_1) \oplus DAT(x_2)) \oplus DAT(x_2) &\leftarrow (DAT(x_1) \cup DAT(x_1)) \cup DAT(x_2) \\ &= DAT(x_1) \cup (DAT(x_1) \cup DAT(x_2)) \\ &\rightarrow DAT(x_1) \oplus (DAT(x_2) \oplus DAT(x_2)) \end{aligned} \tag{8}$$

From (7),(8), we conclude that `concat`($\oplus$) is commutative and associative. Similarly, we also have `join`($\otimes$) commutative and associative. ∎

When propagated upward, depend on the kinds of predecessor nodes and the kinds of connections among nodes, suitable operation (`join` or `concat`) will be applied. `join` is used to generate the DAT of a compound node where the semantic is that all child node are chosen. Whereas, `concat` is used to generate the DAT of a goal node or observable node where the semantic is the selection of one among its successor.

$$DAT(x) \leftarrow \begin{cases} \bigoplus\limits_{\forall \langle g_i, x, p_i \rangle \in E} \left(\texttt{map}_{\langle x,i,p_i \rangle} \textit{DAT}(g_i) \right) & x \text{ is an observable node,} \\ \bigoplus\limits_{\forall \langle c_i, x, 1 \rangle \in E} \textit{DAT}(c_i) & x \text{ is a goal node.} \\ \bigotimes\limits_{\forall \langle x_i, x, p_i \rangle \in E} \textit{DAT}(x_i) & x \text{ is a compound node.} \end{cases} \tag{9}$$

where $\texttt{map}_{\langle x,i,p_i \rangle} \textit{DAT}(x) = \bigcup\limits_{j} \{\langle S_j, p_i \cdot mb_j, 1 - p_i \cdot \overline{rr_j}, \langle x, i \rangle \| T_j \rangle\}$. The operator '$\|$' denotes the string concatenation operation *e.g.,* $a \| \{b, c\} = \{ab, ac\}$.

We assume that there are only one top goal in the goal model. Even though it is not the case in practice, we are always able to introduce a new phantom root goal which is AND-decomposed to all existing top goals. This trick ensures that there is only one root DAT generated. Once the DAT of root node is generated, it is used to calculate the max belief and residual risk of an arbitrary configuration $\mathcal{C}$.

To this end, given an evolutionary goal model eGM with root node is x_0, the following formulae calculate the max belief and residual risk of a configuration $\mathcal{C}$.

$$\begin{aligned} MaxB(\mathcal{C}) &= \max_{\forall \langle S_i, mb_i, rr_i, T_i \rangle \in \textit{SDA}(\mathcal{C})} mb_i \\ RRisk(\mathcal{C}) &= 1 - \sum_{\forall \langle S_i, mb_i, rr_i, T_i \rangle \in \textit{SDA}(\mathcal{C})} \overline{rr_i} \end{aligned} \tag{10}$$

where

$$\forall \langle S_i, mb_i, rr_i, T_i \rangle \in \textit{DAT}(x_0).\mathcal{C} \supseteq S_i \wedge \nexists \langle \cdot, \cdot, \cdot, T_i \rangle \in \textit{SDA}(\mathcal{C}) \Rightarrow \langle S_i, mb_i, rr_i, T_i \rangle \in \textit{SDA}(\mathcal{C})$$

Notice that two or more tuples in an DAT which have a same T_i determine that they are design alternatives fulfilling a same observable evolution possibility. Thus, when calculating residual risk, only one of them is taken into account.

Algo. 1 presents the algorithm that generates *DAT*s for every node in a given evolutionary hypergraph. The algorithm comprise two procedures, `generateDAT` and `initializeDAT`. The later (line 23–34) initializes DATs for every nodes in accordance to Formula 5. This procedure also initializes two data structures, *REACH* and *Q*. The later holds list of *ready-to-process* nodes which have their successors DATs properly generated, whereas the former holds the number of unprocessed child nodes of an arbitrary node. In other words, *REACH[x]* is 0 if and only if x is ready to process. Initially, all leaf nodes are enqueued as they are ready to process.

The former procedure, `generateDAT`, generates DATs of non-leaf nodes by synthesizing successors' DATs. The basic idea of the procedure is as follows. First,

Algorithm 1. Generating DATs for an evolutionary graph.

```
1  procedure generateDAT(G: EvolutionaryHyperGraph)
2  begin
3    initializeDAT(G);
4    while Q ≠ ∅ do
5      n ←dequeue(Q);
6      i ←0;
7      if (REACH[n] = 0) {check whether node is ready, but not visited}
8        REACH[n] ←−1; {mark node visited}
9        for each ⟨ x, n, p⟩ ∈G do {incoming edges of n}
10         i ←i + 1; {i^th incoming edges of n}
11         if n is an observable node then
12           DAT[n] ←concat(n, DAT[n], map(⟨ x, i, p⟩, DAT[x]));
13         else if n is a goal node then
14           DAT[n] ←concat(n, DAT[n], DAT[x]);
15         else
16           DAT[n] ←join(n, DAT[n], DAT[x]);
17       for each ⟨ n, y, p⟩ ∈G do {outgoing edges of n}
18         REACH[y] ←REACH[y] − 1;
19         if REACH[y] = 0 then
20         enqueue(Q, y);
21 end
22
23 procedure initializeDAT(G: EvolutionaryHyperGraph)
24 begin
25   makeQempty();
26   for each x: Node ∈G do
27     if x is leaf then
28       DAT[x] ←{⟨{x} , 1, 0, ∅⟩};
29       REACH[x] ←0;
30       enqueue(Q, x);
31     else
32       DAT[x] ←∅ ;
33       REACH[x] ←|x_in|; {number of incoming edges}
34 end
```

it initializes necessary data structures, *REACH, DAT* and Q (line 3). The `while` loop at line 4 to line 20 processes all *ready* nodes of queue Q. For each dequeued node n of queue Q, the `for` loop at line 9–16 calculates $DAT(n)$ in accordance to Formula 9. The `for` loop at line 17–20 counts down the number of unprocessed children of n's parent nodes by 1 (as n has been processed). If any parent of n is ready (line 19), it is push to queue Q (line 20) for later processing.

Since DATs are maintained at each graph node, if there is any change in the model, the corresponding DATs are then updated. Changes in DATs are propagated to parent node with minimal re-calculation. Algo. 2 describes the algorithm propagating change at an arbitrary node.

Algorithm 2. Updating incrementally DAT.

```
1  procedure updateDAT(G : EvolutionaryHyperGraph, t : node)
2  begin
3    makeQemtpy();
4    enqueue(Q, t);
5    while Q ≠ ∅ do
6      n ←dequeue(Q);
7      i ←0;
8      DAT[n] ←∅ ;
9      for each ⟨ x, n, p ⟩ ∈G do {incoming edges of n}
10       i ←i + 1; {i^th incoming edges of n}
11     if n is an observable node then
12       DAT[n] ←concat(n, DAT[n], map(⟨ x, i, p ⟩, DAT[x]));
13     else if n is a goal node then
14       DAT[n] ←concat(n, DAT[n], DAT[x]);
15     else
16       DAT[n] ←join(n, DAT[n], DAT[x]);
17     for each ⟨ n, y, p ⟩ ∈G do {outgoing edges of n}
18       enqueue(Q, y);
19 end
```

The inputs of Algo. 2 are the evolutionary hypergraph whose nodes' *DAT* have been generated by Algo. 1, and a changed node t. The algorithm initializes a queue Q holding nodes whose *DAT*s need to be updated (line 3). At the beginning, Q holds t. Then, the `while` loop at line 5 sequentially extracts nodes n from Q to update *DAT*(n). Once node n extracted, its *DAT* is reset to empty, then the `for` loop at line 9 re-calculate *DAT*[n] as exactly as Algo. 1 does. After that, the `for` loop at line 17 puts all ancestors of n to Q to further update their *DAT*s.

Theorem 1 *The algorithm 1 always terminates in time polynomial in the number of nodes of the hypegraph and the number of alternative solutions. When terminated, all DATs of all nodes in the input graph are initialized as Formula 9.*

Proof (Theorem 1) Let us consider the first point in the theorem about the polynomial complexity of the algorithm. Algo. 1 has two `for` loops (line 9, 17) nesting inside a `while` loop (line 4). The two `for` loops are proportional to the maximum incoming edges of an arbitrary node. Suppose that the graph has n node, then the maximum incoming edges is $n - 1$. Hence the complexity of these `for` loops are $O(n)$ if we ignore the complexities of `join` and `concat`.

The outside `while` loop terminates when the queue Q empty. At beginning, the initialization at line 3 pushes all leaf nodes to the queue. Later on, only nodes y which $REACH[y]$ equals 0 are enqueued (line 19, 20). When a node y is being processed, its corresponding $REACH[y]$ is marked as -1 (line 8). Thus every node is processed exactly once due to the check at line 7. Hence the complexity of the algorithm is $O(n^2)$.

Algorithm 3. Calculating max belief and residual risk using DAT.

```
1  procedure calcMBRR(DAT : DAT, C : set of Goal, out mb, rr : number)
2  begin
3      trails ←∅ ;
4      totalBelief ←0;
5      mb ←0;
6      for each ⟨ S_i, mb_i, rr_i, T_i ⟩ in DAT do
7        if (C ⊇ S_i and T_i ∉ trails) then
8          mb ←max(mb, mb_i);
9          totalBelief ←totalBelief + (1 − rr_i);
10         trails ←trails ∪ {T_i}
11     rr ←1 − totalBelief;
12 end
```

This values need to be multiplied by the complexity of the `join` and `concat` operations which is bounded by the number of design alternatives in the DAT.

The second point of the theorem is that DATs for all nodes are properly generated when the algorithm terminates. Since `join` and `concat` operators are commutative and associative (see Lemma 1), the DAT calculation of a node by the `for` loop at line 9 obviously follows Formula 9. Furthermore, when a node have all of its children processed, it is immediately put in queue Q. This means that it will be processed sometime later. Therefore, when the algorithm terminates, all nodes have their DAT calculated as Formula 9. ∎

Theorem 2 (Stability of *DAT*) *Each time a node changes, its DAT as well as its ancestors' DATs are updated.*

Proof (Theorem 2) When Algo. 1 is running, once a node x has been visited, $REACH[x]$ is set to -1 (line 8). After that, *DAT*$[x]$ is no longer touch by Algo. 1 because of the precondition check at line 7.

Later on, if users change x *e.g.,* adding new predecessors, Algo. 2 is invoked. Therefore, the *DAT*s of x and its ancestors are updated. ∎

Algo. 3 presents the algorithm calculating max belief and residual risk of a configuration $\mathcal{C}$ using *DAT* of the root node, which generated by Algo. 1. Basically, this algorithm is a translation of Formula 10 in programming language. The algorithm walks though all elements DA_i of the *DAT* (the `for` loop at line 6), checks whether $\mathcal{C}$ supports DA_i, and there is no DA_j which have the same trail with DA_i. For each matched element, the max belief and total belief values are updated (line 8–9). When the `for` loop terminates, the residual risk is the complement of calculated total belief (line 11).

Example 5 (Max Belief, Residual Risk and DAT) We apply the algorithm to the hypergraph denoted in Example 4. Table 2 presents some entries in the DAT of the root goal g_1, for the full $DAT(g_1)$, please refer to Table 3 in the appendix. Next, we apply Formula 10 to calculate the max belief and residual risk of a configuration $\mathcal{C}$= $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$ as follows.

Table 2 Samples in the DAT table of the root node g_1

	Design Alternative (DA)	MB	RR	Trail (T)
1	$\{g_9, g_6, g_7, g_{10}, g_{11}\}$	1.80%	98.20%	$\{\langle ro1, 0\rangle, \langle ro2, 0\rangle\}$
2	$\{\mathbf{g_9}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}\}$	**5.40%**	**94.60%**	$\{\langle \mathbf{ro1}, \mathbf{0}\rangle, \langle \mathbf{ro2}, \mathbf{1}\rangle\}$
3	$\{\mathbf{g_9}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}, \mathbf{g_{16}}, \mathbf{g_{17}}\}$	**5.40%**	**94.60%**	$\{\langle \mathbf{ro1}, \mathbf{0}\rangle, \langle \mathbf{ro2}, \mathbf{1}\rangle\}$
4	$\{\mathbf{g_9}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}, \mathbf{g_{16}}, \mathbf{g_{17}}, \mathbf{g_{19}}, \mathbf{g_{20}}\}$	**4.80%**	**95.20%**	$\{\langle \mathbf{ro1}, \mathbf{0}\rangle, \langle \mathbf{ro2}, \mathbf{2}\rangle\}$
5	$\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	4.80%	95.20%	$\{\langle ro1, 0\rangle, \langle ro2, 2\rangle\}$
6	$\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	4.80%	95.20%	$\{\langle ro1, 0\rangle, \langle ro2, 2\rangle\}$
7	$\{g_9, g_{12}, g_6, g_7, g_{10}, g_{11}\}$	6.30%	93.70%	$\{\langle ro1, 1\rangle, \langle ro2, 0\rangle\}$
8	$\{\mathbf{g_9}, \mathbf{g_{12}}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}\}$	**18.90%**	**81.10%**	$\{\langle \mathbf{ro1}, \mathbf{1}\rangle, \langle \mathbf{ro2}, \mathbf{1}\rangle\}$
9	$\{\mathbf{g_9}, \mathbf{g_{12}}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}, \mathbf{g_{16}}, \mathbf{g_{17}}\}$	**18.90%**	**81.10%**	$\{\langle \mathbf{ro1}, \mathbf{1}\rangle, \langle \mathbf{ro2}, \mathbf{1}\rangle\}$
10	$\{\mathbf{g_9}, \mathbf{g_{12}}, \mathbf{g_6}, \mathbf{g_7}, \mathbf{g_{11}}, \mathbf{g_{15}}, \mathbf{g_{16}}, \mathbf{g_{17}}, \mathbf{g_{19}}, \mathbf{g_{20}}\}$	**16.80%**	**83.20%**	$\{\langle \mathbf{ro1}, \mathbf{1}\rangle, \langle \mathbf{ro2}, \mathbf{2}\rangle\}$
...	...	...	...	...
23	$\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	18.40%	81.60%	$\{\langle ro1, 2\rangle, \langle ro2, 2\rangle\}$
24	$\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	18.40%	81.60%	$\{\langle ro1, 2\rangle, \langle ro2, 2\rangle\}$

Bold lines (2–4, 8–10) are design alternatives supported by configuration $\mathcal{C}$= $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$.

– Identify design alternatives supported by $\mathcal{C}$. Look at Table 2, line 2–4, 8–10 refer to alternatives supported by $\mathcal{C}$.

$$\mathit{SDA}(\mathcal{C}) = \{DA_2, DA_3, DA_4, DA_8, DA_9, DA_{10}\}$$

– Remove alternatives that have duplicated trail (T). Among selected alternatives, DA_2, DA_3 have a same trail, and DA_8, DA_9 also have a same trail. Thus we remove DA_3, DA_9 from $DA(\mathcal{C})$:

$$\mathit{SDA}(\mathcal{C}) = \{DA_2, DA_4, DA_8, DA_{10}\}$$

– Calculate the max belief and residual risk.

$$MaxB(\mathcal{C}) = \max\{5.4\%, 4.8\%, 18.9\%, 16.8\%\} = 18.9\%$$
$$RRisk(\mathcal{C}) = 1 - (1 - 94.6\%) + (1 - 95.2\%) + (1 - 81.1\%) + (1 - 83.2\%) = 54.1\%$$

■

A weak part of the algorithm is the dependence on the number of not-dominated design alternatives. If the design alternatives form a k-ary tree, then the number of leave is about $L \approx \frac{n+1}{k}$, where n is number of alternative task nodes. Then the number of design alternatives would be a power set of L. So in worst case, we need to join two *DAT*s with $2^{L/2}$ elements each, the complexity of `join` thus is $O(2^n)$

Such huge number of alternatives cannot found in practice. Indeed we can simply discard the alternatives that are dominated by other in both max belief and residual risk. The worst case scenario can only materialize in the case of perfect uncertainty: all observable alternatives have exactly the same probability and each design alternative is mutually not compatible with some observable alternative.

The discussion with experts from ATM domains that we carried out in the SecureChange project showed that AND decomposition and observable rules form the bulk of the graph with very few design alternatives. Further, some of the design alternatives distinguish between the vanilla or rich version of a solution so that

one alternative works for both. This further reduces the number of truly alternative configurations.

For example in the goal model of the AMAN system provided by ATM experts, there are 54 goals (36 are leaves), 23 AND-decomposition, but only 6 design alternatives. It means that the complexity of the `join` operator in this real world case is less far than the number of nodes.

9 Related Works

Boyd [6] was one of the first who mentioned the term "enterprise model". He introduced many essentially concepts which later became pervasive in the area of Enterprise Modeling (EM). Vernadat [40] defines EM as the process of building models of the whole or part of an enterprise (e.g., process models, data models, resource models, etc.)

There exists several techniques, frameworks, languages and tools for representation of enterprise and its processes. They includes SADT, Data Flow Diagrams (DFDs), IDEF0 and IDEF [32], activity diagrams in the UML, IEM [22], GRAI/GIM [9], ARIS/EPC [34], UEML [39], CIMOSA [7], GERAM [5]. Besides those, i* modeling framework [41] was introduced to support the modeling of strategic relationships among organizational actors. It provides sets of concepts that complement other approaches for organizational modeling (such as CIMOSA and GERAM) to address the need for expressing complex organizational issues as well as technology ones in an enterprise. However, these models do not cater for the possibility of evolution.

Evolution was considered initially by Ba et al. [3] in their conceptual Enterprise Modeling System (EMS). Their framework automatically builds and executes task-specific models as needed in response to queries from users. The EMS was designed to support strategic decision-making such as predicting the effects of changes in business policies, analyzing possible reactions to internal and external threats, and exploring new business opportunities. In general, the framework tried to address the "what if", "what should we do", and "where should we go"-type of questions. However, this framework was at high abstract level.

Loucopoulous and Kavakli [20] described an approach involving the explicit modeling to express enterprise views in terms of the technical, social and teleological dimensions and their relationship to information system goals. They also illustrated their approach by a case study about Air Traffic Control. In another work [21], they emphasized Enterprise Knowledge Management as the most important factor to address the turbulent changes in many industry sectors. They proposed an enterprise knowledge meta model which consisted of three main elements: 'enterprise goals' which are realized by 'enterprise processes' and implemented by 'information system components'. Each element has its own meta model. The purpose of this enterprise knowledge was the improvement of the enterprise in order to deal with changes. They recommended that enterprise change management should be seen as a process of identifying business goals and relating business processes to these goals. Though the authors did not have any concrete

analysis rather than empirical observations in an industrial application, they confirmed the premise of the key to successfully change is knowledge about "where the enterprise is currently", "where the enterprise wished to be in the future", and "alternative designs" for the desired future state.

Nurcan and others [24, 25, 30] presented the Enterprise Knowledge Development - Change Management Method (EKD-CMM) which was used in the ELEKTRA project [26]. The main goal of EKD-CMM is to provide a framework to understand the way a specific organization works, determine the reasons and requirements for changes, alternatives to address the requirements, and the criteria for the evaluation of the alternatives. The application of the EKD-CMM results in three models, namely the As-Is-Model, the To-Be-Model and the Change Process Model. Also discussed on EKD-CMM, Barrios and Nurcan [4, 23] mentioned a critical need for realistic representations of "what are the current or future business situations", or what should be changed in today enterprise. They provided a roadmap for EKD-CMM as a systematic way to deal with enterprise modeling and transformation. Enterprise objectives (or enterprise goals), processes, and systems are integrated in a single modeling framework using three-layer model.

Other notable works include Dalal et al [8] and Kassem et al [13]. Dalal et al, in their article [8] identified four gaps related to EM methods. Among those, one serious gap is a lack of a formal theory enabling quantitative analysis in the interests of making better business decisions. Our proposal of the quantitative metrics of MaxBelief and Residual Risks directly addresses this issue. In [13], Kassem et al. presented guidelines for the selection of the right modeling method, and proposed a methodology for Enterprise Modeling. They found that the selection should be a function of: the purpose, the ease of communication between stakeholders, the characteristic of the modeling environment and the characteristic of the modeling technique itself. The methodology is based on the combined use of IDEF0 and Dependency Structure Matrix (DSM) to produce functional requirements for the collaborative software. This methodology hence is capable to understand complex interactions, facilitate the management of change, and create a shared vision of business processes. However, the authors did not focus on any further analysis for change management.

10 Conclusion

In this work we have addressed the issues of modeling evolutions of enterprise systems. In particular, we focused on potential evolutions which can be foreseen, but it is not sure that these evolutions do actually happen. We called this problem *managing known unknown*.

Our proposed approach introduces the notion of *evolution rules*, comprising of observable and controllable rules, as a mechanism for handling this phenomenon. The uncertainty of evolution is expressed using probability values represent expertise beliefs on the occurrence of evolutions, which is accounted using game-theoretic approach. Additionally, we provided a brief discussion about the graphical notion for representing these rules, which has been validated with ATM experts. Furthermore, based on that belief, we introduce two quantitative metrics to

measure the level of usefulness of design alternatives. This provides a quantitative analysis that supports decision makers in deciding which is the optimal configuration for their enterprise.

To exemplify our approach, we discussed a case study in ATM domain. This case study is about deploying AMAN to the working position of ATCOs, and its applicability aspects have been subject to a number of sessions with ATM experts. From that case study, we have drawn several examples to explain our idea, and also to show the promising applicability of our approach.

Our future work is geared towards providing an interaction protocol with stakeholders that allow a simple elicitation and validation of the probability estimates. A promising avenue seems to use the Analytic Hierarchy Process (AHP) method for a first cut description and then use our game semantics to validate and refine them.

References

1. C. Anderson. The long tail. *Wired*, October 2004.
2. A. I. Antón and C. Potts. Functional paleontology: The evolution of user-visible system services. *IEEE Transactions on Software Engineering*, 29(2):151–166, 2003.
3. S. Ba, A. B. Whinston, and K. R. Lang. An enterprise modeling approach to organizational decision support. In *Proceedings of the twenty-eighth annual Hawaii International Conference on System Sciences*, pages 312–320, 1995.
4. J. Barrios and S. Nurcan. Model driven architectures for enterprise information systems. In *Proceedings of the 16th Conference On Advanced Information Systems Engineering*, pages 3–19, 2004.
5. P. Bernus. GERAM: Generalised enterprise reference architecture and methodology. version 1.6.3, March 1999.
6. D. Boyd. A new management technique. *Enterprise models: Industrial Management Review*, 8(1), 1966.
7. CIMOSA. Computer integrated manufacturing open system architecture: A primer on key concepts, purpose and business value. Online primer, Accessed April 2009.
8. N. P. Dalal, W. J. Kolarik, and E. Sivaraman. Toward an integrated framework for modeling enterprise processes. *Commun. ACM*, 47(3):83–87, March 2004.
9. G. Dougmeingts, Y. Ducq, B. Vallespir, and S. Kleinhans. Production management and enterprise modelling. *Comput. Ind.*, 42:245–263, July 2000.
10. A. Elberse. Should you invest in the long tail? *Harvard Business Review*, 2008.
11. J. L. Fiadeiro. On the challenge of engineering socio-technical systems. In *Software-Intensive Systems and New Computing Paradigms*, volume 5380 of *Lecture Notes in Computer Science*, pages 80–91. Springer-Verlag, 2008.
12. J. Hassine, J. Rilling, J. Hewitt, and R. Dssouli. Change impact analysis for requirement evolution using use case maps. In *IWPSE '05*, 2005.
13. M. Kassem, N. N. Dawood, and D. Mitchell. A structured methodology for enterprise modeling: a case study for modeling the operation of a british organization. *Journal of Information Technology in Construction*, 16:381–410, 2011.
14. C. F. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4):493–509, 1999.
15. W. Lam and M. Loomes. Requirements evolution in the midst of environmental change: a managed approach. In *CSMR '98*, 1998.

16. M. LaMantia, Y. Cai, A. MacCormack, and J. Rusnak. Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases. In *Proc. of WICSA '08*, pages 83–92, 2008.
17. M. Lehman. On understanding laws, evolution and conservation in the large program life cycle. *J. of Sys. and Soft.*, 1(3):213 –221, 1980.
18. M. Lehman. Programs, life cycles, and laws of software evolution. *Proc. IEEE 68*, 9:1060 –1076, September 1980.
19. L. Lin, S. Prowell, and J. Poore. The impact of requirements changes on specifications and state machines. *SP&E*, 39(6):573–610, 2009.
20. P. Loucopoulos and E. V. Kavakli. Enterprise modelling and the teleological approach to requirements engineering. *International Journal of Intelligent and Cooperative Information Systems*, 4(1):44–79, 1995.
21. P. Loucopoulos and E. V. Kavakli. Enterprise knowledge management and conceptual modelling. In *Selected Papers from the Symposium on Conceptual Modeling, Current Issues and Future Directions*, pages 123–143, London, UK, 1999. Springer-Verlag.
22. K. Mertins and R. Jochem. Integrated enterprise modeling: method and tool. *ACM SIGOIS Bulletin*, 18(2), 1997.
23. S. Nurcan and J. Barrios. Enterprise knowledge and information system modelling in an evolving environment. In *International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE'03)*, 2003.
24. S. Nurcan, J. Barrios, G. Grosz, and C. Rolland. Change process modelling using the ekd change management method. In *Proceedings of 7th European Conference on Information Systems, ECIS'99*, pages 513–529, June 23-25 1999.
25. S. Nurcan and C. Rolland. A multi-method for defining the organizational change. *A multi-method for defining the organizational change*, pages 61–82, 2003.
26. Project ELEKTRA. ELEKTRA: Enhanced Learning Experience and Knowledge TRAnsfer. http://www.elektra-project.org, Retrieved April 4 2011.
27. Project PROTEUS. Deliverable 1.3: Meeting the challenge of chainging requirements. Technical report, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.
28. Project SECURECHANGE. Deliverable 1.1: Description of the scenarios and their requirements. http://securechange.eu/content/deliverables, 2010.
29. R. Ravichandar, J. Arthur, S. Bohner, and D. Tegarden. Improving change tolerance through capabilities-based design: an empirical analysis. *J. of Soft. Maintenance and Evolution: Research and Practice*, 20(2):135–170, 2008.
30. C. Rolland, S. Nurcan, and G. Grosz. Enterprise knowledge development: the process view. *Information & Management*, 36(3):165 – 184, 1999.
31. J. Rooksby, M. Rouncefield, and I. Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Computer Supported Cooperative Work*, 18(5-6):559–580, 2009.
32. D. Ross and K. Schoman. Structured analsysis for requirements definition. *IEEE Transactions on Software Engineering*, 3(1):69–84, 1977.
33. A. Russo, B. Nuseibeh, and J. Kramer. Restructuring requirements specifications. In *IEE Proceedings: Software*, volume 146, pages 44 – 53, 1999.
34. A.-W. Scheer. *ARIS–business process modeling*. Heidelberg, German: Springer-Verlag, 2000.
35. G. Shafer, V. Vovk, and R. Chychyla. How to base probability theory on perfect-information games. *BEATCS*, 100:115 – 148, February 2010.
36. P. Soffer. Scope analysis: identifying the impact of changes in business process models. *J. of Soft. Process: Improvement and Practice*, 10(4):393–402, 2005.

37. L. M. S. Tran. Dealing with known unknowns: A goal-based approach for understanding complex systems evolution. Technical report, University of Trento, 2011.
38. L. M. S. Tran and F. Massacci. Dealing with known unknowns: Towards a game-theoretic foundation for software requirement evolution. In *Proceedings of the 23th Conference On Advanced Information Systems Engineering*, pages 62–76, 2011.
39. F. Vernadat. UEML: Towards a unified enterprise modelling language. *International Journal of Production Research*, 40(17), 2002.
40. F. B. Vernadat. *Enterprise Modeling and Integration Principles and Applications*. Chapman and Hall Publisher, 1996.
41. E. Yu. Strategic modelling for enterprise integration, 1999.
42. D. Zowghi and R. Offen. A logical framework for modeling and reasoning about the evolution of requirements. *ICRE '97*, 1997.

Appendix

Table 3 Full DAT table of the root node g_1

Design Alternative (DA)	MB	RR	Trail (T)
1 $\{g_9, g_6, g_7, g_{10}, g_{11}\}$	1.80%	98.20%	$\{\langle ro1, 0\rangle, \langle ro2, 0\rangle\}$
2 $\{g_9, g_6, g_7, g_{11}, g_{15}\}$	5.40%	94.60%	$\{\langle ro1, 0\rangle, \langle ro2, 1\rangle\}$
3 $\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}\}$	5.40%	94.60%	$\{\langle ro1, 0\rangle, \langle ro2, 1\rangle\}$
4 $\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$	4.80%	95.20%	$\{\langle ro1, 0\rangle, \langle ro2, 2\rangle\}$
5 $\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	4.80%	95.20%	$\{\langle ro1, 0\rangle, \langle ro2, 2\rangle\}$
6 $\{g_9, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	4.80%	95.20%	$\{\langle ro1, 0\rangle, \langle ro2, 2\rangle\}$
7 $\{g_9, g_{12}, g_6, g_7, g_{10}, g_{11}\}$	6.30%	93.70%	$\{\langle ro1, 1\rangle, \langle ro2, 0\rangle\}$
8 $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}\}$	18.90%	81.10%	$\{\langle ro1, 1\rangle, \langle ro2, 1\rangle\}$
9 $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}\}$	18.90%	81.10%	$\{\langle ro1, 1\rangle, \langle ro2, 1\rangle\}$
10 $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
11 $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
12 $\{g_9, g_{12}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
13 $\{g_9, g_{13}, g_6, g_7, g_{10}, g_{11}\}$	6.30%	93.70%	$\{\langle ro1, 1\rangle, \langle ro2, 0\rangle\}$
14 $\{g_9, g_{13}, g_6, g_7, g_{11}, g_{15}\}$	18.90%	81.10%	$\{\langle ro1, 1\rangle, \langle ro2, 1\rangle\}$
15 $\{g_9, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}\}$	18.90%	81.10%	$\{\langle ro1, 1\rangle, \langle ro2, 1\rangle\}$
16 $\{g_9, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
17 $\{g_9, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
18 $\{g_9, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	16.80%	83.20%	$\{\langle ro1, 1\rangle, \langle ro2, 2\rangle\}$
19 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{10}, g_{11}\}$	6.90%	93.10%	$\{\langle ro1, 2\rangle, \langle ro2, 0\rangle\}$
20 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}\}$	20.70%	79.30%	$\{\langle ro1, 2\rangle, \langle ro2, 1\rangle\}$
21 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}\}$	20.70%	79.30%	$\{\langle ro1, 2\rangle, \langle ro2, 1\rangle\}$
22 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}\}$	18.40%	81.60%	$\{\langle ro1, 2\rangle, \langle ro2, 2\rangle\}$
23 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{21}\}$	18.40%	81.60%	$\{\langle ro1, 2\rangle, \langle ro2, 2\rangle\}$
24 $\{g_9, g_{12}, g_{13}, g_6, g_7, g_{11}, g_{15}, g_{16}, g_{17}, g_{19}, g_{20}, g_{21}\}$	18.40%	81.60%	$\{\langle ro1, 2\rangle, \langle ro2, 2\rangle\}$

APPENDIX F

Understanding How to Manage Requirements Evolution: An Experience in the Air Traffic Domain *

F. Paci[1], M.S. Tran[1], F. Massacci[1], D. Nagaraj[1], and A. Tedeschi[2]

[1] DISI - University of Trento,
{paci, tran, massacci,nagaraj}@disi.unitn.it
[2] Deep Blue
alessandra.tedeschi@dblue.it

Abstract. [**Context/Motivation**] Requirements evolution is still a major problem in practice. Continuously changing requirements make the traceability of requirements difficult and the monitoring of requirements unreliable.[**Question/ Problem**] In this paper we tackle the problem of modeling and reasoning about requirements evolution. Our research objectives are to gain in-depth understanding of the change management process adopted in a real industrial setting, and investigate how the approach to model and reason on requirements evolution previously proposed in [15] can facilitate the industrial change management process. [**Principal Ideas/Results**]This paper presents the results of the qualitative user study we have conducted in the air traffic management (ATM) domain where change management is still an open issue [13]. The user study has involved eight ATM domain experts in the assessment of the approach to requirement evolution. [**Contribution**] The results from the study demonstrated the usefulness of the approach to requirements evolution in the ATM industrial setting. Furthermore, important lessons were learned about the key aspects of the approach to evolution and about the aspects to be taken into account during research methodology's design.

Keywords: requirement engineering, evolution, change management, user study, air traffic domain

1 Introduction

The requirements for any non-trivial system will inevitably change, not only after the system has been built, but also during the process of implementing the system. Changes make the traceability of requirements hard and the monitoring of requirements unreliable: requirements management is difficult, time-consuming and error-prone when done manually. Thus, change management is a major problem in practice. However, very few empirical studies [6, 16, 9, 10, 3] about requirements evolution have been reported.

This paper presents the results of a user study conducted about modeling and reasoning on requirements evolution. The study has been carried out using a qualitative research approach. Qualitative research aims to investigate and understand social and

* Work partly supported by the projects EU-FP7-IP-SecureChange, EU-FP7-NoE-NESSOS.

cultural phenomena in the context where they exist. The study is mainly built on semi-structured interviews with a high degree of discussion completed with focus groups meetings.

The objectives of the user study are twofold: a) gain in-depth understanding of the change management process adopted in a real industrial setting, and b) investigate the role that an approach to model and reason on requirements evolution (previously proposed in [15]) can play in such process. The approach consists of a representation of an evolving requirement model as a set of *controllable* and *observable rules* and a reasoning technique based on two metrics called *maximal belief* and *residual risk* that intuitively measure the usefulness of a model element (or a set of elements) after evolution.

As industrial setting we have considered the air traffic management (ATM) domain for two main reasons. First, the ATM systems are complex and critical systems that are going through significant structural, operational and cultural changes as planned by the EU Single European Sky ATM Research(SESAR) Initiative [1]. Second, change management is still an open issue in the ATM domain: the need of system engineering techniques to support change management is well recognized [13]. For the user study, we have focused on the change management process followed by Air Navigation Service Providers to introduce at operational level a new decision supporting tool, the AMAN. The AMAN suggests to the Air Traffic Controllers (ATCOs) an optimal arrival sequence of aircrafts and provides support in establishing the optimal aircraft approach route.

The user study has involved twelve participants of which four were requirement analysts who form the research team while the other were ATM domain experts who have been involved in the change management process to introduce the AMAN. It also provided useful insights into the weaknesses and advantages of our approach to requirement evolution's modeling and reasoning. The user study demonstrated the usefulness of the approach to requirements evolution in the change management process. The reasoning supported by the approach to requirements evolution has been considered by the ATM experts as a powerful decision-support tool that allows one to select the optimal design which is resilient to future changes in a requirements model. Moreover, we have learnt important lessons about the aspects to be taken to into account when a research methodology is designed.

The paper is organized as follows. Section 2 presents the related work. Section 3 gives an overview of the approach to requirements evolution which has been validated during the workshop. Section 4 presents the research methodology. Section 5 summarizes the results of the user study while Section 6 presents the relevant findings and concludes the paper.

2 Related Work

Several studies have been conducted to understand the challenging aspects of requirements engineering. Curtis et al. [5] published the first significant field study exploring how requirements and design decisions are made. Crucial issues reported by the participants of the study were conflicting requirements and communication breakdowns.

Chatzoglou et al. [4] conducted a study about the management of requirements capture and analysis (RCA) process, to investigate the presence of significant differences between projects developed by different people and organisations. The study identifies as main challenging aspect the lack of adoption of a methodology during the RCA process.

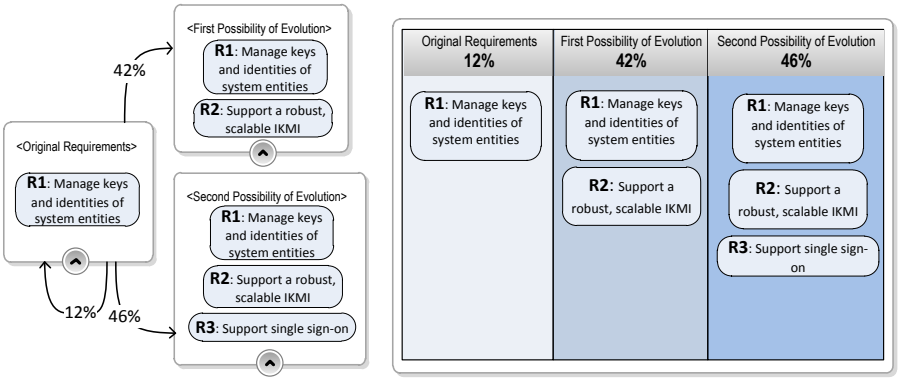
Several studies discuss the challenges associated with requirements modeling. Lubars et al. [11] summarized the findings of a field study involving ten organizations, focusing on the problems of vaguely stated requirements and requirements prioritization. Other issues were the unclear relation of performance requirements with parts of dataflow/control flow specifications. In [2], Bertolino et al. provided an approach for validating a domain model with experts ensuring the model captures properly the intended domain knowledge. Another relevant work include the one of Maiden et al. [12] who have conducted workshops with stakeholders using a concurrent engineering process focusing on system goal modeling to model the future system.

The closest works to ours include the studies by Herrmann et al. [6], Welsh and Sawyer [16], Kamsties et al. [9], Karlsson et al. [10], and Carlshamre et al. [3]. Herrmann et al. described a process for capturing delta requirements (requirements describing enhancements) deducing the level of detail to be maintained in the existing system. In another study, Welsh and Sawyer use requirement traceability information to handle requirement changes in Dynamic Adaptive System's (DAS). They also propose an extension to i* strategic rationale models to aid changing a DAS. Kamsties et al. and Carlshamre et al. have investigated the role of requirements interdependencies when new requirements need to be implemented for next system releases. Kamsties et al. summarized the results of a workshop on requirements engineering held with practitioners from ten small and medium enterprises. The main result from the workshop was that new requirements implementation can cause unpredictable interactions with requirements that are already implemented. Carlshamre et al. conducted a survey of five different companies that has shown that visualization of requirement interdependencies is efficient in identification of salient characteristics of requirements. Karlsson et al. pointed the importance to have methods to cope with changing requirements.

Compared to above studies on changing requirements, the user study we have conducted does not only aim at understanding how requirements evolution is handled in an industrial setting (the ATM domain) but it also focused on validating our concrete approach to deal with requirements evolution.

3 An Approach to Requirements Evolution

This section gives an overview of our approach [15] to manage the evolution of requirements. Requirements evolution refers to changes in the requirements of a system. In our work we consider two categories of evolution: *controllable* and *observable*. *Controllable evolutions* are under the control of system designers who intentionally change the system design in order to fulfill high level requirements proposed by stakeholders. In other words, they are designers' moves to identify design alternatives to implement a system. *Observable evolutions*, instead, represent evolutions which are not under the control of the designer, but that can be somehow detected when they happened or whose future likelihood can be estimated with a certain confidence by the stakeholders. To support decision making process, our proposed approach [15] incorporates these kinds



(a) Tree-like representation (b) Swimlane-based representation

Fig. 1. Visualization of evolution rules.

of evolutions into requirement models in terms of *evolution rules*. There are two kinds of rule: *controllable rule* and *observable rule*. They are formally defined as follows:

$$\textit{Controllable rule:} \qquad r_c = \bigcup_{i=1..n} \left\{ Before \xrightarrow{*} After_i \right\} \tag{1}$$

$$\textit{Observable rule:} \qquad r_o = \bigcup_{i=1..n} \left\{ Before \xrightarrow{p_i} After_i \right\} \tag{2}$$

where $Before$ denotes the before-evolution requirements model and $After_i$ denotes one of the possible requirements model to which $Before$ can evolve to, and $p_i \in [0..1]$ is the probability that $Before$ will evolve to $After_i$. We denote p_i as *evolution probability*.

Example 1. Let assume that the requirements model for the AMAN introduction (denoted as RM_1) states that the requirement R_1 *-Manage keys and identities of system entities (human, software, devices etc)* needs to be satisfied. RM_1 is associated with a set of observable rules r_o which specify RM_1 can evolve to a requirements model RM_2 with a probability of 42%, or to a requirements model RM_3 with probability 46% or it can remain unchanged with probability 12%. RM_2 is a requirements model that requires the satisfaction of the requirements $R1$*-Manage keys and identities of system entities (human, software, devices etc)* and R_2*-Support a robust IKMI that can be scaled up to large number of application and users*; while RM_3 is a requirement model which requires the satisfaction of the requirements $R1$, $R2$, and $R3$*-Single Sign-On (SSO) support*.

The observable evolution rules for the example above can be graphically represented in two ways: *tree-like* and *swimlane-based* representations. Fig. 1(a) illustrates the former representation. The root node is the $Before$ state which is directly linked to the $After_i$ states. Links are labeled with evolution probabilities p_i. Fig. 1(b), describes the swimlane-based representation where $Before$ state is on the left side, and all the possible $After_i$ states are located on the right, next to the $Before$ state. The evolution probabilities are specified on swimlanes' tops.

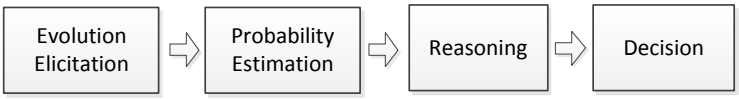


Fig. 2. Requirements Engineering process for Change/Evolution Management

The elicitation of evolution rules is the first step of the change management illustrated in Fig. 2. The process consists of four major steps, namely *Evolution elicitation*, *Probability Estimation*, *Reasoning*, and *Decision* [14].

STEP 1 **Evolution Elicitation.** The goal of this key step is to extract evolution rules from a description in natural language of the possible requirements evolutions. In order to do that, first, statements about changes are identified. Then, changes are clustered into mutually exclusive groups. Then, each group will represent an $After_i$ requirement model.

STEP 2 **Probability Estimation.** Based on the evolution rules identified in the previous step, this step relies on methods like Analytic Hierarchy Process (AHP) to identify the evolution probability for each possible evolution $After_i$ of an observable evolution rule. The probabilities are then validated with stakeholders(and/or domain experts) using a game-theoretic approach [15] to ensure that they are meaningful.

STEP 3 **Reasoning.** Based on the representation of an evolving requirement model as a set of controllable and observable rules, it is possible to compute two quantitative metrics called maximal belief and residual risk that intuitively measure the usefulness of a model element (or a set of elements) after evolution. In fact, the maximal belief tells whether a design alternative is useful after evolution, while residual risk quantifies if a design alternative is no longer useful.

STEP 4 **Decision.** Decision makers take the analysis' outcome to choose the optimal design for the next development phases. Depending on the kind of analysis, different "optimal" criterion can be applied. For instance, a selection criterion for the analysis is: "higher max belief and lower residual risk".

4 Research Methodology

Many different assessment methodologies can be used for validation with domain experts. For this study, we have mainly used qualitative techniques. *Qualitative research* consists of an application of various methods of collecting information, mainly through focus groups and interviews. Interviews are commonplace techniques where domain experts are asked questions by an interviewer in order to gain domain knowledge and it is the most widely used method of finding out what users want [8]. Focus groups bring together a cross-section of stakeholders in an informal discussion group format. The user study has involved twelve participants having different background and roles as summarized in Table 2. P1, P2, P3 and P4 played the role of *requirement analyst* and *observer*. Requirement analysts were responsible for the organization of the user study and for the analysis of the results. The observers were responsible to record the meetings and take notes during the execution of the study. The other participants were ATM domain experts who participated to the semi-structured validation of the proposed approach to model and reason on requirements evolution.

To conduct the user study we have designed a research method that consists of three main steps: *User Study Design*, *User Study Execution*, and *Analysis of Results*.

Table 1. Research Questions and Success Criteria

	Research Questions(RQ) and Success Criteria(SC)	
RQ1	*How is evolution managed in a fully realistic and complex setting such as ATM domain?*	
	SC1	The phases of the change management process adopted by Air Navigation Service Providers are identified. For each phase of the process the participants and the artifacts used to support the phase are identified.
RQ2	*Can the approach to model and reason on requirements evolution be applied to the change management process adopted by Air Navigation Service Providers?*	
	SC2	The approach to model and reason on requirements evolution can support and facilitate the change management process by by Air Navigation Service Providers;
	SC3	The approach to model and reason on requirements evolution is sufficiently expressive to capture the evolution in the ATM domain;
	SC4	The approach to model and reason on requirements evolution can be easily understood by ATM domain experts

4.1 User Study Design

The first step of the research study was to identify the research questions to be addressed and the success criteria to evaluate the results obtained from the user study(see Table 1). The main objective of the user study is to understand the process to handle change adopted by Air Service Navigation Providers and to investigate the applicability of the approach to model and reason on evolution to the different phases of the process.

Once identified the research questions, the ATM domain experts were selected. Eight domain experts were involved in the user study: P5 has worked in several projects related to the ATM domain and has a good knowledge of requirements engineering domain, P6 has been an air traffic controller for several years, P7 and P8 has worked in operational projects regarding the development and the management of ATM systems, P11 and P12 are air traffic controllers and P12 and P13 are Human Factors and Safety experts in the ATM domain.

Since ATM domain experts were not very familiar with requirements engineering and do not have knowledge about the approach to model and reason on requirements evolution, it was necessary to prepare training material in form of presentations. The training material includes presentations about

- main concepts and analysis techniques relevant for change management in requirements engineering domain
- the approach to model and reason on requirements evolution (see Section 3).

The introduction of the AMAN was used as illustrative example through all the training materials since the ATM domain experts were familiar with this scenario. Also an interview guide containing a list of questions to lead the discussions during the interviews was prepared.

Table 2. Participants Background

Participants	Role	Position	Education	Years of Professional Experience
$P1$	RA,OBS	PostDoc	PhD	2
$P2$	RA,OBS	PostDoc	PhD	4
$P3$	RA,OBS	PhD	MSc	1
$P4$	RA,OBS	Full Professor	PhD	> 10
$P5$	DE	Consultant	MSc	20
$P6$	DE	Consultant	PhD	> 5
$P7$	DE	ICT Manager	MSc	> 10
$P8$	DE	SyS Admin	MSc	> 10
$P9$	DE	ATCO	MSc	> 10
$P10$	DE	ATCO	MSc	> 10
$P11$	DE	Human Factors & Safety Expert	PhD	> 10
$P12$	DE	Human Factors & Safety Expert	PhD	> 10

The table gives an overview of the role and the background of each of the participants of the experiment. In the table OBS = OBServer, RA = Requirement Analyst, DE = Domain Expert, ATCO = Air Traffic COntroller, SyS Admin = System Administrator.

The feedback collection from the ATM experts was carried out through semi-structured interviews and focus groups and by recording the meetings for later analysis.

4.2 User Study Execution

Table 3 outlines the organization of our study. The first column denotes the meeting type, which can be preliminary meeting (PM), internal working session (IWS) or workshop (WS). The second column specifies when the meeting took place. The third column lists the participants of the meetings. Finally, the last column indicates what happened during the meeting.

The study was organized into two separated workshops: one held in April (WS1), and the other one in June 2011 (WS2). Before conducting these two workshops, one preliminary meeting (PM1) was held among requirement analysts and observers to identify the research questions of the study, the success criteria, and the strategy to be followed during the study. In between the meetings, there were internal working sessions (IWS1 and IWS2) where requirement analysts prepared relevant materials for the study, including case study description and models, presentations, and other training materials such as the interview guide. These documents were distributed to the domain experts before the workshops in order to being able to collect feedbacks from them.

The two workshops were divided into two sessions: Designing session and Validation session. The former lasted the first day of WS1: requirement analysts discussed with domain experts P5, P6, P11 and P12 about prepared training material (*e.g.,* case study, presentation slides) to get their feedbacks. Then the training material were modified based on domain experts's feedbacks. The Validation session was held during the

Table 3. User Study Organization

Meeting	Date	Participants	Contents
PM1	March, 2011	RA, OBS	Participants identify the research questions of the user study, and the success criteria. Further more, the strategy to be followed to conduct the study is sketched.
IWS1	March, 2011	RA, OBS, DE	RAs,OBSs and DEs define and agree upon the research design.
IWS2	March, 2011	RA	RAs prepared the training materials (tutorials about requirements engineering, requirements evolution, interview guide. These documents were distributed to all the DEs.
WS1 (day 1)	April, 2011	RA, OBS, DE	RA present the training material to domain experts. Domain experts give feedback to enrich each of discussed items, and to improve the way they are presented.
WS1 (day 2)	April, 2011	RA, OBS, DE	RA gives an introductory talk about ideas of evolution. Next, RA presents research objects. During each of presentation, DEs are interviewed to collect feedbacks about the change management process adopted by ANSPs.
IWS3	May 2011	RA,OBS	The RAs analyze feedbacks from WS1 and prepare for WS2.
WS2	June, 2011	RA, OBS, DE	Participants continue the validation work. DEs elaborate more on the approach to model and analyze requirements evolution.
IWS4	August, 2011	RA, OBS,	RAs analyze the feedbacks collected during the two workshops.

The table shows the organization of the user study. In this table PM = Preliminary Meeting, IWS = Internal Working Session, WS = WorkShop.

second day of the first workshop and during the second workshop. In this session, requirements analysts gave presentations about the proposed approach to model and reason on evolution. The domain experts were asked whether the approach would make sense or useful based on their experience. The domain experts also reported the existing process to change management adopted by Air Navigation Service Providers.

4.3 Results Analysis

The qualitative data collected in the form of video recording were transcripted to understand and review the conversations between the ATM experts and the requirement analysts. The tool used to perform qualitative data analysis is NVivo 9 [7]. Using NVivo, the transcription is coded (in terms of qualitative analysis) to distinguish between the various topics of discussion and points of feedback. Different codes and their descriptions are reported in Table 4. The coding was further analyzed to emerge with evidence

for the defined success criteria. On the various transcripts, the code percentage coverage was performed to display the percentage of occurrence of a code in comparison with other codes. The percentage coverage diagram reported in Figure 3 shows that the conversations related to operational procedures and change management procedures have maximum coverage on the transcription.

Table 4. Description of codes used.

Code	Description
AMAN	The specific discussion points related to AMAN which covers the factors considered as constraints for AMAN, the expectations of the experts from AMAN and the procedures associated with AMAN.
Change Management	The discussions on the change management procedure which includes the steps of Brain Storming, Risk Assessment, Simulation, Human Verification, Type of Change, Evolution, Change Management Procedure and Constraints for change management.
Functionality	The specific functionality mentioned in accordance to the role of either the Controller or the Coordinator.
Goals	The discussions related to goal refinement and goal sharing concepts in i*-based requirements modeling language.
Operational Procedure	The operational procedures followed by Air Traffic Controllers in the real situation.
Organizational Role	The composition of organizational roles in ENAV.
Feedback on Graphical Representation	The feedback obtained on the graphical representation of evolution presented by the requirement engineers.
Behavior	The behavior observed apart from the discussion points as silence, indirect answer and not interested.

4.4 Threats to Validity

We consider threats to construct, internal, external and conclusion validity [17] as relevant for our user study.

– **Construct Validity**:Threats to construct validity are related to decide to which extent what was to be measured was actually measured.The main construct validity threat in this study regards the design of the measurement instrument: are the questions formulated so that the interviews answer our research questions? Our main measurement instrument is the interview guide which includes the questions to be asked. Three requirement analysts have constructed it by analyzing the research questions and creating sub-questions. The list of questions have been reviewed by the consultants to check for completeness and consistency; therefore we believe

that the interview guide is accurate. Moreover, to reduce this threat we have used a number of information sources and data collection techniques.

- **Internal Validity**:Internal validity is an inductive estimate of the degree to which conclusions about causal relationships can be made (e.g. cause and effect), based on the measures used, the research setting, and the whole research design. A threat to internal validity in our case study was related to communication issues with the ATM experts. First of all there was a communication gap between the requirements analysts and the ATM domain experts: certain concepts have a different meaning for them. A second issue was related to language barriers: the ATM expert had to provide feedbacks not in their mother tongue language. For these reasons, some useful feedbacks might not have been given to the requirement analysts.
- **External Validity**:External validity concerns the extent to which the (internally valid) results of a study can be held to be true for other cases, for example to different people, places or times. Since we have conducted a qualitative user study, our objective was not to do any generalization of the results to other contexts, but understand the change management process adopted when a new tool is introduced in complex and critical systems such as air traffic management systems and to investigate the applicability of the approach to model and reason on evolution. To generalized the findings of this user study we are planning to consider other complex systems in different application domains.
- **Conclusion Validity**:This threat is concerned with the degree to which conclusions reached are justified. Our conclusions are based on the remarks and comments given by the ATM experts. The video recorded discussions between the ATM experts and requirement analysts have been analyzed to provide evidence in accordance to the real context of the ATM domain. To have an effective discussion with the ATM experts, the requirement analysts also had a prior meeting with them to confirm the protocol for the discussion.

5 Analysis of the Results

This section summarizes the feedbacks that were collected during the two workshops WS1 and WS2 and analysis of the feedbacks by the requirement analysts.

5.1 Feedbacks During the Experiments Execution

On the applicability to the ATM domain of the approach to model and reason on requirements evolution the participants P5 and P12 have pointed out that the approach could be very useful during the initial phase of the change management process adopted by ANSPs. The approach can help managers, technicians and controllers to understand the implications and impacts of a proposed change. P5 and P12 suggested that Evolution Elicitation and Probability Estimation might be useful to identify the alternative operational requirements associated with a proposed change. The Reasoning phase instead can be used to support the decision makers in identifying the best solution at operational level to be implemented.

A challenge in applying the approach to model and reason on requirements evolution was reported by participants P6 and P9. They argued that due to the complexity of

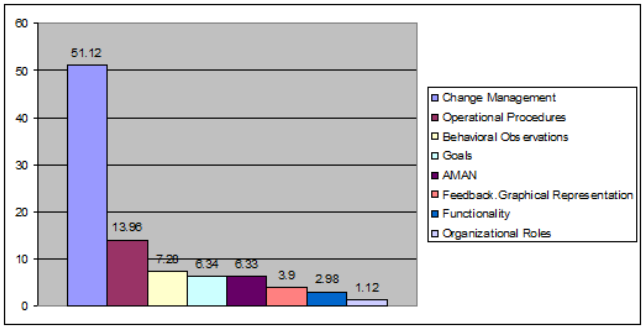


Fig. 3. Codes Coverage

ATM systems, it is very difficult to identify all possible evolutions in advance and to predict the probability of evolutions. P6 and P9 also remarked that it is possible to identify certain evolutions only during the change impact analysis phase. Thus, the ATM experts suggested that an incremental approach should be adopted to identify all possible evolutions for a given before-evolution requirements model. Moreover, participant P12 has pointed out that qualitative metrics such as risk severity are already used to classify a change, and therefore new metrics should be related to those already in use.

On the graphical representation of evolution rules, all participants have agreed that they prefer the tree-like representation illustrated in Figure 1(a) rather than the swim lane representation in Figure 1(b).

Moreover, participant P13 has expressed a positive opinion about the applicability of the graphical representation to capture the evolution in ATM, P6 and P9 instead were not totally confident about that.

5.2 Observations from Requirement Analysts

Some of the main observations made by the requirement analysts are reported in what follows.

- Build requirement models with the experts triggers many useful discussions with and among the experts, and help reveal inconsistencies and mistakes in the models.
- More feedbacks could have been collected if the ATM experts would have applied the approach to model and reason on requirements evolution.
- There was a communication gap between requirement analysts and domain experts regarding the terminology used in the study. ATM experts should have been encouraged to provide also written feedbacks to reduce the language barriers.
- It would have been better to involve all the different stakeholders - air traffic controllers, managers, software engineers- who participate in the change management process adopted by ANSPs.
- Having internal working sessions with the domain experts before the workshops is key for collecting useful feedbacks from the ATM experts.

5.3 Evaluation with respect to the Success Criteria

In this section we evaluate the performance of the modeling and reasoning approach to requirements evolution with respect to the success criteria identified in section 4.1.

SC1 : *The phases of the change management process adopted by Air Navigation Service Providers (ANSP) are identified. For each phase of the process the participants and the artifacts used to support the phase are identified.*

As attested by the graph illustrated in Figure 3, during the WS1 and WS2 workshops, the requirement analysts gain a deep-understanding of the change management process adopted by ANSPs. The process mainly consists of the following phases:

1. **Brainstorming**. This phase aims at exploring the different options for responding to a proposed change and evaluating their feasibility. Air traffic controllers, technical, and managers try to identify the new high level operational requirements and the different possible alternative operational and technical procedures. The operational requirements and procedures are represented as influence diagrams [13].
2. **Risk Assessment**. The objective of this phase is to identify the nature of the proposed change. If there is no impact at operational level the change is minor and can be directly implemented without performing any simulation or training for the air traffic controllers. On the contrary, if simulation is needed, the change is a major change.
3. **Fast Time Simulation**. The aim of this phase is to assess the impact of change on the operational procedures by performing computer-based simulation.
4. **Real Time Simulation**. This phase evaluates the impact of the change with humans in a real environment. Humans can propose minor changes to be implemented.

SC2 : *The approach to model and reason on evolution can support and facilitate the change management process adopted by Air Navigation Service Providers*.
The participants P5 and P12 have pointed out that currently, in the whole ATM domain an increasing interest is devoted to methodologies and processes supporting and documenting the decision making activities within the SESAR Programme. They have mentioned that main open issues are change management and the need of a formal methodology to trace and assess the introduction of new operational concepts and their impact on ATM Key Performance Areas. P5 and P12 have reported that now the different phases of the change process adopted by Air Navigation Service Providers are supported by influence diagrams that allow to trace strategic objectives to operational solution and that allow to perform what if analysis to understand the impact of a proposed change. P6 and P9 suggested that the Evolution Elicitation and Probability Estimation might be useful during the brainstorming phase to identify the alternative operational requirements associated with a proposed change. The Reasoning phase instead can be used to support the decision makers in identifying the best solution at operational level to be implemented.

The comments of P5 and P12, thus, indicate that the approach to model and reason on evolution can support managers and controllers during each phase of the change process.

SC3: *The approach to model and reason on requirements evolution is sufficiently expressive to capture the evolution in the ATM domain*.

The feedbacks provided by the ATM expert P5 and P9 during the second workshop indicate that the representation of evolution rules can be applied to model evolution in the ATM domain. The experts pointed out that since ATM systems are complex systems it might be difficult to predict all possible evolutions and represent them as after requirement models. Thus, they suggested to adopt an incremental approach to identify the possible evolutions. Moreover, the experts reported that the probability of evolution is difficult to determine. They also pointed out that qualitative parameters are associated with evolution in the ATM domain rather than the probability of evolution.
SC4:*The approach to model and reason on requirements evolution can be easily understood by ATM domain experts*.
The graphical representations for evolution rules illustrated in Figures 1(a) and Figure 1(b) were presented to the ATM experts during WS1 the first workshop. The ATM experts were asked which graphical representation they prefer and they preferred the tree-like representation. They were also asked if the find intuitive the graphical representation and they agreed that the graphical representation could be easily understand by them. This answer is also supported by the fact that the domain experts suggested modifications to the requirement models that were the after models of an observable evolution rule, explained their rationale, or asked relevant questions about some detail in the models. This indicates the graphical representation and the before and after-evolution requirements models were comprehensible for the domain experts.

6 Lessons Learnt and Conclusions

This paper has presented the results of a qualitative user study about requirements evolution. The objectives of the study were to gain in-depth understanding of the change management process adopted by Air Navigation Service Providers when a new tool such as the AMAN is introduced, and to investigate the role that the approach to model and reason on requirements evolution can play in such process. The study was mainly built on semi-structured interviews with a high degree of discussion between the requirement analysts and ATM domain experts, and on focus groups meetings. This approach has allowed the requirement analysts to understand the change management process adopted by Air Navigation Service Providers.

The user study also has provided useful insights into the weaknesses and advantages of our approach to requirement evolution's modeling and reasoning. Moreover, we have learnt important lessons about the aspects to consider during research design. We summarized the main findings in what follows.

6.1 Results Related to the approach to Requirements Evolution

A lesson learnt from the user study is that the most challenging step of our approach to requirement evolution is Evolution Elicitation. Regarding the Evolution Elicitation phase, not all after requirement models for an observable rule can be foreseen in advance. Estimating the probability of evolution of after requirements models is not a trivial process. The tree-like representation for evolution rules is easy to understand but its intuitiveness can be undermined if the before and after evolution requirements

models are too complex. The intuitiveness depends on the requirement language used to represent before and after evolution models.

The Probability Estimation and the Reasoning phase are instead the key phases of the approach to requirements evolution. The Probability Estimation gives the ability to "translate" qualitative metrics that are typically used by stakeholders to classify evolution into quantitative metrics that enable the Reasoning phase. The Reasoning phase has turned out to be a powerful decision-support tool since it allows to select the optimal design which is resilient to changes in the requirements model.

6.2 Experiences from Using a Qualitative Research Approach

During the execution of the user study, we have understood that several aspect can influence the feedback collection from domain experts. The selection of domain experts strongly influence the relevance of feedbacks collected and the satisfaction of the success criteria chosen for the user study. In the user study, the domain experts selected had a different background and so we were able to collect feedbacks about the approach to requirement evolution from different perspectives.

Another important aspect to take into account is the potential communication gap between the research team and the domain experts. Research team and domain experts might use same terms with different meanings that can lead to misunderstandings and to provide wrong or unrelated feedbacks. Thus, it is required to establish a common language between the research team and the domain experts before the user study execution, or to have a "mediator" who reformulates the questions of the research team for the domain experts and who reformulates the domain experts' feedback for the research team. Moreover, the level of engagement of the domain experts depends by two main factors: the means to provide feedbacks and the language in which such feedbacks need to be provided. Different ways to collect feedbacks not only verbal one need to be supported. The most effective one must be the one in which the domain experts can discuss in their mother tongue language and then provide written feedback in English.

To collect more insightful feedbacks, hands-on sessions where the domain experts apply/ use the artefacts under validation should be included in the validation session execution.

Acknowledgement

We would like also to thank DeepBlue and ATM experts for participating in this study. We would like to thank Professor John Mylopoulos, and colleagues in the ATHENA research group at University of Trento (UNITN) for their scientific contribution, and Dr. Alberto Battocchi (UNITN) for acting as an observer.

References

1. EUROCONTROL ATM Strategy for the Years 2000+ Executive Summary, 2003.
2. A. Bertolino, G. D. Angelis, A. D. Sandro, and A. Sabetta. Is my model right? let me ask the expert. *Journal of Systems and Software*, 2011.

3. P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. och Dag. An industrial survey of requirements interdependencies in software product release planning. 2001.
4. P. D. Chatzoglou. Factors affecting completion of the requirements capture stage of projects with different characteristics. *Information & Software Technology*, 1997.
5. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. 1988.
6. A. Herrmann, A. Wallnöfer, and B. Paech. Specifying changes only — a case study on delta requirements. REFSQ '09, pages 45–58, Berlin, Heidelberg. Springer-Verlag.
7. H. Jemmott. Using nvivo for qualitative data analysis. *Analysis*, 1(February):7–7, 2008.
8. F. J. F. (Jr) and T. W. Mangione. *Standardised Survey Interviewing*. Springer Publishing Company, Incorporated, 1990.
9. E. Kamsties, K. Hörnmann, and M. Schlich. Requirements engineering in small and medium enterprises. In *Proc. Conf. on European Industrial Requirements Engineering*, 1998.
10. L. Karlsson, A. Dahlstedt, B. Regnell, J. Nattochdag, and A. Persson. Requirements engineering challenges in market-driven software development – An interview study with practitioners. *Information and Software Technology*, 49(6):588–604, June 2007.
11. M. Lubars, C. Potts, and C. Richter. A review of the state of the practice in requirements modelling. In *Proceedings of IEEE International Symposium on Requirements Engineering, RE'93*, pages 2–14, San Diego, California, 1993. IEEE Computer Society Press.
12. N. A. M. Maiden, C. Ncube, and J. Lockerbie. Inventing requirements: Experiences with an airport operations system. 2008.
13. H. K. Rober Graham, Nadine Pilon and P. Ravenhill. Performance framework and influence model in atm. In *Digital Avionics Systems Conference(DASC)*, 2009.
14. L. M. S. Tran. Dealing with known unknowns: A goal-based approach for understanding complex systems evolution. Technical report, University of Trento, 2011.
15. L. M. S. Tran and F. Massacci. Dealing with known unknowns: Towards a game-theoretic foundation for software requirement evolution. In *CAiSE*, pages 62–76, 2011.
16. K. Welsh and P. Sawyer. Requirements tracing to support change in dynamically adaptive systems. REFSQ '09, pages 59–73, Berlin, Heidelberg, 2009. Springer-Verlag.
17. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

APPENDIX G

Quick fix generation for DSMLs

Ábel Hegedüs*, Ákos Horváth*, István Ráth*, Moisés Castelo Branco† and Dániel Varró*
*Budapest University of Technology and Economics
Budapest, Hungary
Email: {hegedusa,ahorvath,rath,varro}@mit.bme.hu
†University of Waterloo
Waterloo, Ontario, Canada
Email: mcbranco@gsd.uwaterloo.ca

***Abstract*—Domain-specific modeling languages (DSML) proved to be an important asset in creating powerful design tools for domain experts. Although these tools are capable of preserving the syntax-correctness of models even during free-hand editing, they often lack the ability of maintaining model consistency for complex language-specific constraints. Hence, there is a need for a tool-level automatism to assist DSML users in resolving consistency violation problems. In this paper, we describe an approach for the automatic generation of quick fixes for DSMLs, taking a set of domain-specific constraints and model manipulation policies as input. The computation relies on state-space exploration techniques to find sequences of operations that decrease the number of inconsistencies. Our approach is illustrated using a BPMN case study, and it is evaluated by several experiments to show its feasibility and performance.**

I. Introduction

In model-driven software engineering, domain-specific modeling (visual) languages (*DSML*s) provide a popular rapid prototyping and generative technique to increase design reusability, speed up the development process and improve overall quality by raising the level of abstraction from software specific details to the problem domain itself. Today, DSMLs are deployed in many development toolchains to aid both software designers and domain experts, in order to integrate deep domain knowledge at an early design phase.

While many standard DSMLs are readily available (such as AUTOSAR for automotive software design, or process modeling languages such as BPMN [1] for service-oriented systems), custom modeling environments are developed in-house by many organizations to tailor development tools and code generators to their specific needs. This *language engineering* process relies on frameworks such as the Eclipse Modeling Framework (EMF [2]) or MetaEdit+ [3] that provide powerful tools for defining DSMLs and to automatically generate textual or graphical editors for creating domain models.

As domain-specific models are abstract and thus highly compact representations of the system-under-design, a key issue (that arises in both standardized and custom-made modeling environments) is *inconsistency management*. Inconsistencies are violations of the well-formedness and correctness rules of the language that may correspond to design errors, or violations of the company's policies and best practices. While domain-specific editors are usually capable of ensuring that elementary editing operations preserve syntactic correctness (by e.g. *syntax-driven editing*), most DSMLs include additional language-specific consistency rules that must also be checked.

In the current state-of-the-art of modeling tools, inconsistency management of complex rules focuses primarily on the *detection* of inconsistency rule violations, facilitated by dedicated constraint evaluators that take e.g. an OCL [4] expression and generate code that continuously scans the models and reports problematic model elements to the user. However, the *resolution* of these violations is mostly a manual task: the user may manually alter the model to eliminate the violations, but this can be a very challenging problem due to the complexity of the language or the concrete model. As a result, manually fixing a violation of one inconsistency rule may introduce new violations of other rules.

In *programming* languages, the concept of *quick fixes* (also called error correction, code completion) is a very popular feature of integrated development environments such as Eclipse or Microsoft Visual Studio, which aids programmers in quickly repairing problematic source code segments. This feature is deeply integrated into the programming environment and it can be invoked any time for a detected violation, giving the developer a list of fixing actions that can be applied instantly.

In the current paper, we propose to adapt this concept to domain-specific modeling languages. Our aim is to provide a domain-independent framework (that is applicable for a wide range of DSMLs), which can efficiently compute complex fixing action sequences even when multiple, overlapping inconsistency rule violations are present in the model. To capture inconsistency rules of a DSML, we use graph patterns that define declarative structural constraints. Our technique uses graph transformation rules to specify elementary fix operations (policies). These operations are automatically combined by a structural constraint solving algorithm that relies on heuristics-driven state-space exploration to find quick fix sequences efficiently. As a result, our approach provides high-level extensibility that allows both the language engineer and the end user to extend the library of supported inconsistency rules and fixing policies.

The rest of the paper is structured as follows. Section II introduces the problem of model editing for DSMLs and describes our BPMN case study used for illustration throughout the paper. In Section III, we give a more precise definition on quick fixes for DSMLs, describe the process for generating

them and specify the architecture used for implementation. In Section IV, we show the application of the approach on the BPMN case study, while Section V contains our evaluation of the application. Finally, related work is discussed in Section VI and Section VII concludes the paper.

II. Case study: business process modeling

In this paper, we use the Business Process Model And Notation (BPMN [1]) as an illustrative case study. BPMN is a well-known and widely used standard, flowchart-like notation system for specifying business processes. BPMN supports the scoped modeling of both control and data flow; for control flow, activities, events, gateways (conditional decision, fork-join) may be used while data flow may be facilitated between activities and artefacts (e.g. data objects). All elements of a process may be organized into pools and swimlanes to create a structural breakdown according to e.g. organizational or functional rules (scopes).

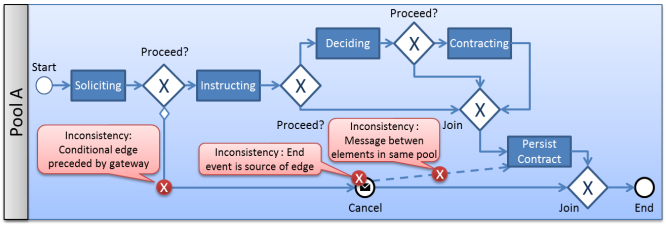


Fig. 1. Example BPMN process

Currently, there is a variety of BPMN-capable tools available (e.g. The MEGA Suite, Tibco Business Studio, or SAP NetWeaver BPM) that all use the standardized graphical notation (depicted in Figure 1), where activities (tasks) are represented by rectangles, events by circles, gateways by diamonds (rhombi) and sequence flow by arrows – inconsistencies are indicated by red circles). However, in our examples we use a graph-like *abstract syntax representation* that indicates the types and names of each BPMN node more explicitly in order to ease understanding.

Unfortunately, these tools do not address automated inconsistency resolution. For example, while the popular BPMN editor included in the Eclipse SOA Tools Platform suite [5] is able to detect a number of simple inconsistency rule violations, it does not guide the user in any way how to fix them.

Example 1 (Quick fix in a BPMN context) We give an intuitive example to clarify the concept of *quick fixes* applied in a domain-specific modeling context. *Inconsistency rules* in BPMN may describe process patterns that should not appear according to the standard (or custom design policies), e.g. because they would lead to incorrect behavior (when the process is executed). For instance, as message sending provides communication between different processes (which are modeled as pools), *messages between elements belonging to the same pool* are discouraged.

When an instance of such an inconsistency is detected in a BPMN process model (e.g. a given message is to be sent between two activities that belong to the same pool, as illustrated by the dashed arrow in Figure 1), the Eclipse editor places an error feedback marker in the model to indicate the location of the error. The user can only attempt to correct the error by a manual process, starting from guidance given by the (ad-hoc) textual report. One may attempt to remove the erroneous model configuration by performing a sequence of elementary operations (such as deleting an element, or altering its location), and then re-running the validation on the model to check whether the attempt was successful.

We aim to automate this lengthy and error-prone process by a *quick fix generator*. Given the error marker and a set of elementary fix operations (policies), the generator performs the try-and-test search (without permanently altering the model) and presents the possible solutions (if any) to the user. In the small example above, the generator may find that the violation can be removed by either deleting the message or moving the receiver to a separate pool, and present a list of these solutions to the user, where the (sequence of) correcting operations can be quickly executed in one compound operation, restoring the correctness of the model.

Challenges of quick fix generation

While simple fixing actions (such as deleting the ill-configured message) can be provided (in a hard-wired manner) by the programmer of the violation detection component (as available in e.g. the EMF Validation Framework [6]), our fix generator is a *more generic* solution as it can effectively deal with situations where (i) multiple errors are present in the model, possibly affecting an overlapping set of elements, and (ii) the fixing of individual errors may interfere with each other generating intermediate violations.

From the end-user perspective, our approach addresses the following *challenges/requirements*:

- *quick feedback to the user*: all fixing proposals should be calculated quickly to keep the interactive nature of the modeling process intact (if no proposal can be calculated, the user may wish to continue looking for a solution which takes more time).
- *offer the best fixes to the user*: the inconsistency resolution might have many solutions. The tool should pick the *best* options and present only those to the user, by discarding too complicated or "dangerous" ones (i.e. high number of modifications on a large part of the model).
- *keep model changes at a minimum*: all offered fix proposals should use conservative manipulation sequences that keep most of the model intact, in order to maintain the user's feeling of being in control of model manipulation.
- *support for local and global scope for fixes*: inconsistency rule violations are usually local in the sense that their scope is defined in terms of a few interconnected model elements. While a model may have many erroneous configurations scattered, a fixing proposal generator should allow the user to select the context to which solutions will be generated.

- *extensibility*: the supported library of inconsistency rules and (elementary) fixing policies should be easily extensible by the end users as well. This is a key feature for enforcing customized design standards that organizations or individuals can develop and tailor to their needs.

III. Problem Formalization

A. Definition of Quick Fixes for DSMLs

First, we define the components of a DSML, which are foundational concepts in our approach (depicted in Figure 2).

Definition 1 The *metamodel* MM for a DSML includes the set of model element *types* of the domain, their *attributes* and *relationships* between model elements. A model conforming to the metamodel is called an *instance model* M.

Definition 2 A *query* q over a model M represents constraints that have to be fulfilled by a part of the instance model $M_0 \subseteq M$. Given an instance model M and a query q, a submodel that fulfills the query is called an injective *match* $m : q \mapsto M_0$ (or shortly, $q \overset{m}{\mapsto} M_0$), where M_0 is the context of the match m (denoted as $ctx(m)$).

Definition 3 *Inconsistency rules* r describe situations which should not appear in instance models and they are defined as queries over an instance model. The set of inconsistency rules defined for a DSML is denoted by R.

Definition 4 An *operation* o is a pair of $o = (p, \vec{s})$ with a set of symbolic parameters p, a sequence of elemental model manipulation steps $\vec{s}$, which specify how the model is modified by adding, removing or changing parts of it similar to syntax-directed editing. The set of all operations for a DSML is denoted by O.

Definition 5 The *execution* of the operation o on M_{old} $(M_{old}, b) \overset{o}{\rightarrow} M_{new}$ modifies the model M_{old} according to the elemental steps, based on the bindings b from symbolic parameters to model elements, resulting in model M_{new}.

Definition 6 The *DSML* is a triplet $DSML = (MM, R, O)$ containing the metamodel, the set of inconsistency rules and operations defined for the language.

Next, the above definitions are used for specifying the concepts used in our quick fix generation approach.

Definition 7 An inconsistency rule r is *violated* by an instance model M if there exists at least one match (*violation*, v) $r \overset{v}{\mapsto} M_0$, where $M_0 \subseteq M$. The set of violations that contain a given model element e is denoted by $V_e(M) = \{v | e \in ctx(v)\}$, and finally, the set of all violations in M by $V(M)$.

Definition 8 An instance model M of metamodel MM, is *inconsistent* if one or more inconsistency rules are violated by (a part of) M, formally $\exists r \in R, \exists v \in V(M) : r \overset{v}{\mapsto} M$. The *total number of violations* in M for all inconsistency rules is denoted by $|V(M)|$, while the *number of local violations* for a given model element $e \in M$ is denoted by $|V_e(M)|$.

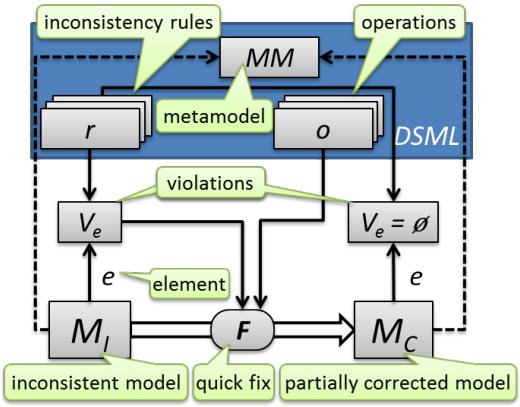


Fig. 2. Quick fixing an inconsistent model

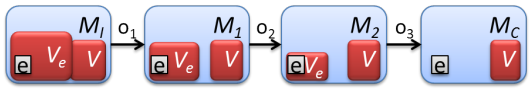


Fig. 3. Application of a Quick fix

Quick fixes were defined informally as a sequence of model manipulation operations, which change an inconsistent model in a way to eliminate constraint violation instances.

Definition 9 A *quick fix* $M_I \overset{F}{\Rightarrow} M_C = (M_i, b_1) \overset{o_1}{\rightarrow} M_1, (M_1, b_2) \overset{o_2}{\rightarrow} M_2, \ldots, (M_{n-1}, b_n) \overset{o_n}{\rightarrow} M_C$ for a model element e is an ordered sequence of operations executed on an inconsistent model M_I, resulting in a *partially corrected model* M_C, with the following conditions:

- $\exists v_I \in V(M_I) : e \in ctx(v_I)$; There exists a violation v_I in the incosistent model M_I, where the model element e is in the context of v_I.
- $\nexists v_C \in V(M_C) : e \in ctx(v_C)$; There is no violation v_C in the partially corrected model M_C, where the model element e is in the context of v_C.
- $|V(M_I)| > |V(M_C)|$; F decreases the total number of violations in the model

Figure 3 illustrates how the application of the operations in a quick fix affects the instance model by eliminating the violations step-by-step. Initially, there are several model elements included in $V_e(M_I)$ in the inconsistent model M_I. After applying the first operation o_1 from the quick fix, the resulting M_1 where some of the violations may be fixed already. By executing the rest of the operations o_2, o_3 in the quick fix, the final model M_C contains no violations for the selected element e. It is important to note that not all violations in the model are eliminated by the quick fix, only those that contained the selected model element e. However, the total number of violations $|V(M_I)|$ decreases.

In order to map our approach to a given specific modeling environment, the generic definitions (such as query and operation) are mapped to the well-known formal technique of graph

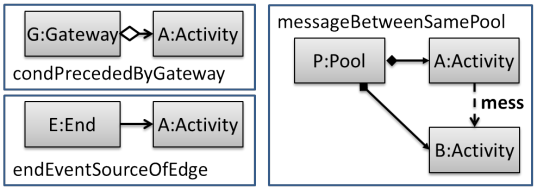


Fig. 4. Inconsistency rules for BPMN

transformation [7] with available, extensive tool support.

B. Graph transformation as modeling formalism

In this paper, both the metamodel and the instance models are specified by (typed and attributed) graphs, with special *instance of* relations between elements and types.

Queries (e.g. inconsistency rules) are defined using *graph patterns* (or graph constraints) [8] including structural, attribute, nested and negated constraints (the last describing cases where the match of a pattern is not valid) and logic composition operators. Querying is performed by graph *pattern matching* [9] that returns matches in the form of graphs.

Graph transformation (GT) [7] provides a declarative language for defining the manipulation of graph models by means of GT rules. At GT rule consists of (i) a left-hand side (LHS), (ii) a right-hand side (RHS) graph, and (iii) arbitrary number of negative application conditions (NAC) attached to the LHS. Model manipulation is carried out by replacing a matching of the LHS in the model by an image of the RHS. This is performed in two phases. In the pattern matching phase, matchings of the LHS are sought in the model and the absence of the graph structures of NACs is checked and ensured. In the updating phase, the selected matching parts are modified based on the difference of LHS and RHS.

The foundation of our approach is similar with [10]–[12] in using graph transformation based techniques for specifying inconsistency rules and model generation.

C. Quick fixes for BPMN

The metamodel of the BPMN language is defined in EMF and is incorporated in the Eclipse BPMN Modeler tool [5]. We used this metamodel for specifying inconsistency rules and model manipulation operations for the language.[1]

1) Inconsistency rules: Figure 4 shows three of such rules as graph pattern using a simple graphical notation. Model elements are depicted with rectangles and relationships with arrows, while the name of the element and the its type are separated by colons.

Conditional Edge Preceded By Gateway: This inconsistency rule (*condPrecededByGateway*) specifies the situation where a *Gateway G* is the source of a conditional sequence edge (depicted as a continuous arrow with a empty diamond source end) with the target an arbitrary activity *A*. If there are two elements in the model, which can be matched to *G* and *A* (respecting the type restriction), then it is a violation. In the case study, such a violation exists on the edge between the first *Proceed?* gateway and the *Cancel* event (see Figure 1).

End Event Source of Edge: This inconsistency rule (*endEventSourceOfEdge*) specifies the situation where an *End* event (*E*) is the source of a sequence edge (with the target an arbitrary activity *A*). If there are two elements in the model, which can be matched to *E* and *A* (respecting the type restriction), then it is a violation (e.g. the sequence edge starting from *Cancel* in the case study as illustrated in Figure 1).

Message Between Elements in Same Pool: The inconsistency rule *messageBetweenSamePool* describes the situation outlined in Example 1. Activities *A* and *B* are both elements in *P* (as defined by the arrow with the diamond shaped end). A violation exists, if three elements matching *A*, *B* and *P* can be found in the model (e.g. the message edge leading from *Cancel* to *Persist Contract* in the case study, see Figure 1).

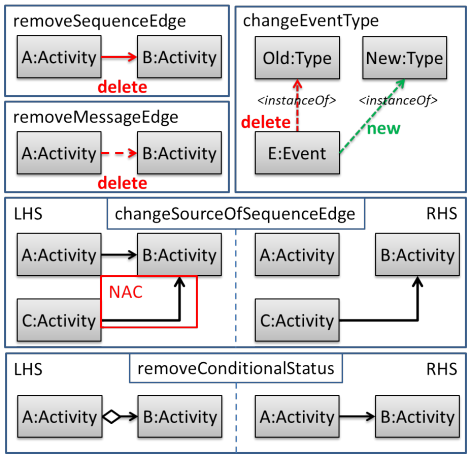


Fig. 5. Operation definitions for BPMN

2) Operation definitions: Operations for manipulating BPMN models can be defined specifically for each metamodel type (e.g. create Parallel Gateway) or generically to decrease the total number of operations (e.g. create element with type). Figure 5 shows five operations for various modifications using graph transformation rules. Negative application conditions are depicted with red rectangles. When possible (the upper three in Figure 5), the LHS and RHS are merged and model parts removed by the operation are annotated with *delete*, while parts that are created are annotated with *new*.

The rules *removeSequenceEdge* and *removeMessageEdge* remove an existing sequence or message edge from between two activities, respectively, while *changeEventType* changes the type of an event element to the given type, depicted as replacing the *instanceOf* relation (e.g. the type of an event is changed from End to Start).

The *changeSourceOfSequenceEdge* moves the source end of a sequence edge between activity *A* and *B*, so that the new source will be activity *C*. It also restricts the application by stating that *C* must not have an existing sequence edge from the same element (*NAC*). Finally, the *removeConditionalStatus* rule removes the conditional status from a sequence edge

[1]The full list of inconsistency rules and operations can be found at http://viatra.inf.mit.bme.hu/publications/quickfix.

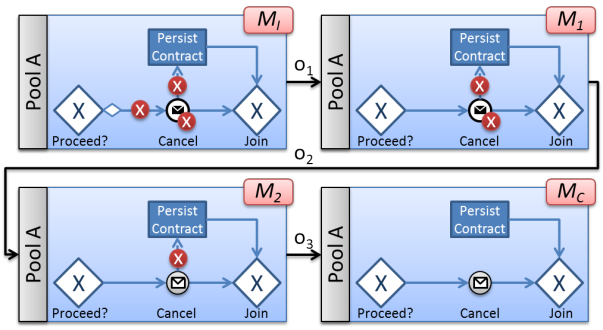


Fig. 6. Application of a fix

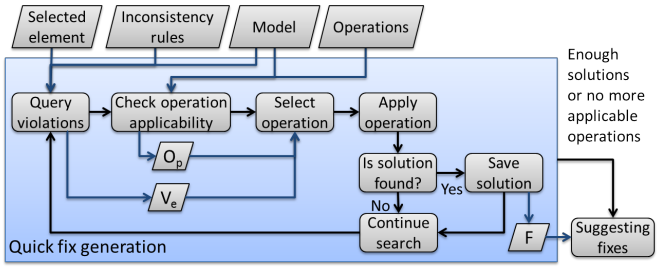


Fig. 7. Overview of the Quick fix generation

between activity *A* and *B*.

3) Application of a quick fix: The quick fixes (generated by the techniques detailed in Section IV) should contain enough information to allow deterministic application over the model (to a degree that violations are eliminated in all cases). A selected quick fix is applied to the model by taking each operation in order and execute it with the stored input bindings (illustrated in Figure 6).

The application starts from the inconsistent model M_I (upper left) and first removes the conditional attribute from the sequence edge between *Proceed?* and *Cancel* resulting in M_1 (upper right). Next, the type of event *Cancel* is changed to intermediate (since it has incoming edges), thus leading to M_2 (lower left). Finally, the message edge between *Cancel* and *Persist Contract* is removed (since they are in the same pool). In the resulting model M_C (lower right) no violations remain on *Cancel*.

IV. Generation of Quick Fixes

A. Constraint satisfaction problem over models

Quick fixes are generated directly on the DSML by using a state-space exploration approach capable of solving structural constraint satisfaction problems over models, also called CSP(M) [13]. In CSP(M), problems are described by: (i) an *initial model* representing the starting point of the problem, (ii) *goals* that must be met by a solution model defined as graph pattern and finally, – as a distinctive feature from traditional CSP approaches – (3) *labeling rules*, which explicitly define permitted operations as graph transformation rules.

In CSP(M) a state is represented by the underlying model – the starting state is the initial model – and a transition between states is an application of a labeling (GT) rule. To explore this state space the CSP(M) solver uses guided traversal algorithms [14] to find a valid solution model that satisfies all goals and minimize the visited states (*effective selection* challenge). Based on the traversal algorithm (which supports both backtracking and cycle detection) the output of the solver can be a single or multiple solution models and the sequence of labeling rules applied to achieve these models.

In order to generate quick fixes using the CSP(M) approach, we encoded the negated inconsistency rules as goals, the allowed operations as labeling rules and used the model from the editor directly as the initial model. This way the CSP(M) solver tries to find a model that satisfies each inconsistency rule using the allowed operations. The main advantage of using the CSP(M) approach is that it allows to define the quick fix problem directly over the DSML model and it does not need any mapping or abstraction to other mathematical domain as used in similar model generation approaches [15], [16].

However, it is important to mention that as the CSP(M) framework is extendable with custom solver algorithms, we modified its solver algorithm by restricting (1) the application of operations and (2) the solution checking to the violations of the selected model element. This is important in order to *support local fixing scopes*. Furthermore, the solver supports both (i) breadth-first and (ii) depth-first search, and (iii) parameterizable limits on solution length and number of alternative solutions. We defined priorities for the operations, which are taken into consideration during the iteration, thus higher priority operations are executed first. By giving higher priority to more conservative operations, the *conservativity* challenge is addressed, since solutions containing these operations are explored before others.

B. Quick Fix Generation Process

The process of quick fix generation depicted in Figure 7 consists of the following steps:

1) *Query violations* The quick fix generation starts with selecting an element e in the model to work as a scope for inconsistency rules. Next, all the violations that include e are queried from the model for each inconsistency rule, initializing the set of violations $V_e(M)$.
2) *Check operation applicability* First, all operations are checked for executability (i.e. whether they can be executed at all) and executable operations are collected in a list Op.
3) *Select operation* The state space exploration then iterates through Op and checks the possible input parameter bindings against elements in the matches for violations in $V_e(M)$ (e.g. a sequence edge violating *endEventSourceOfEdge* will be part of the possible inputs of *removeSequenceEdge*, see Figure 4 and Figure 5).
4) *Apply operation* If it finds a matching input, then the operation is applied to the model with the given input

resulting in a new model state.

5) *Is solution found?* The new model state may be a correct model, this is checked by re-executing the query against the model again to get $V_e(M)$. If $V_e(M)$ is empty, then the total number of violations and violations on elements in the original $V_e(M)$ are checked as well.
   a) *Save solution* When a valid quick fix is found, the trace (with the executed operations and input bindings) is saved to a solution list. Quick fix generation terminates once a predefined number of solutions are found.
   b) *Continue search* If the new model state is not a correct model or further solutions are required, the next applicable operation is selected. The state-space exploration terminates if there is no applicable operation within the limited search space.
6) *Suggesting fixes* The solutions are then suggested for inspection to the user who may choose one quick fix to be applied on the model. If no solutions were found, this information is displayed instead.

It is important that the set of inconsistency rules and operations are easily extensible by the end users (*extensibility* challenge). In our approach, these definitions are not hard-coded into the solver and can be modified using the graph transformation formalism. The CSP(M) framework also supports dynamic handling of inconsistency rules and operations, e.g. to generate solutions for different subsets of operations.

C. Implementation architecture

We implemented our quick fix generation approach using the VIATRA2 model transformation framework [17], which provides metamodeling capabilities and supports model transformations based on the concepts of graph transformations and abstract state machines. Its incremental pattern matcher is used as a powerful query engine [18].

The state space exploration part of the approach is executed by the constraint satisfaction engine presented in [13], where operations and inconsistency rules are used in solving a constraint satisfaction problem over the input model. BPMN processes can be developed in the Eclipse BPMN modeler tool [5]. Since both the modeler tool and VIATRA2 are Eclipse technologies, we could seamlessly integrate the quick fix generator to be usable directly from the modeling tool, just like quick fixes work in integrated development environments.

V. EVALUATION

A. BPMN models used for evaluation

We evaluated the approach for scalability on two real BPMN projects, obtained from an industrial partner from the banking sector. One project is a corporate customer registering workflow, composed of five processes and approximately 250 model activities in total. The other project is a corporate procurement workflow, composed of three processes and around 70 model activities. The projects were selected among others available from the partner by the following criteria: 1 – they can be converted to the Eclipse BPMN Modeler editor with minor changes, since they were originally modeled in another tool; 2 – they allow to explore all the errors described for the case study by containing the necessary modeling scenarios, such as multiple pools and message flows; 3 – all eight BPMN models are classified as typical real-life BPMN processes [19]. The name and size of the different processes are shown in Figure 8.

	Name	Elements	Edges	Subprocesses	Pools
Centralized Register	Macro	16	12	4	1
	Soliciting	20	25	0	1
	Instructing	40	45	2	1
	Deciding	9	10	0	1
	Contracting	36	43	2	1
Procurement	Delivery	8	8	0	2
	Purchase Request	13	13	0	3
	Purchase Order	14	15	0	1

Fig. 8. Processes in the case study

B. Evaluation environment and method

The evaluation was carried out by adding inconsistencies to each process and running the quick fix generation approach independently. We performed measurements[2] multiple times for each test case including different total and local number of inconsistencies in the model.

The measurement of a given test case was done as follows: the inconsistent BPMN model is loaded into VIATRA2, the inconsistency rules and operations are added to the framework, the quick fix engine is initialized and time measurement is started. Next, the quick fix engine looks for three different solutions and gathers them in a list, once it is done the time measurement is stopped. Finally, the results are saved and the framework is disposed to return the environment to the initial state.

C. Evaluation of results

The table in Figure 9 shows the results of our measurements using the case study models. For each model we measured the performance for the given number of total and local inconsistencies. For each case, we measured the number of visited states and the time of quick fix generation. Finally, measurement results are given with the mean values along with deviations.

We made the following observations based on the results from the different models:

One local violation ($\#1-5, 8, 9$)*:* In these cases fixing is possible in a reasonable time even during editing, since the quick fix generation takes less than 4 seconds in all cases except $\#3$, where finding three different solutions takes 15 seconds. Although the deviation of runtime is significant in some cases (50% for $\#11$), it causes only a few seconds longer runtime.

[2]All measurements were carried out on a computer with Intel Centrino Duo 1.66 GHz processor, 3 GB DDR2 memory, Windows 7 Professional 32 bit, Eclipse 3.6.1, EMF 2.6.1, BPMN 1.2, VIATRA2 3.2 (SVN version)

#	Model	|V(M)|	|V_e(M)|	T [ms]	D_T	S	D_S
1	Macro	5	1	1 330	0,10	205	0,00
2	Deciding	3	1	550	0,16	113	0,00
3	Deciding	3	1	14 759	0,04	7 034	0,04
4	Soliciting	4	1	3 982	0,03	608	0,00
5	Contracting	9	1	1 169	0,04	158	0,00
6	Instructing	13	2	46 035	0,05	11 325	0,03
7	Instructing	13	3	165 695	0,02	41 512	0,01
8	Delivery	1	1	432	0,05	109	0,00
9	PurchaseRequest	3	1	508	0,06	106	0,00
10	PurchaseOrder	5	2	30 722	0,04	14 109	0,01
11	PurchaseOrder	5	2	1 480	0,49	405	0,25

Fig. 9. Evaluation results ($|V(M_I)|$: total number of violations, $|V_e(M)|$: max. no. of violations per element, T: time [ms], D_T: standard deviation of time, S: no. of visited states, D_S: standard deviation of visited states)

Locality $(\#3, 5)$*:* The higher number of local violations for the selected element leads to slower fix generation, while the total number of violations in the model does not affect performance. Generating quick fixes for one violation in case $\#3$, where there are only 3 violations, is 15 times slower than for case $\#5$, which has 9 violations. This is a direct consequence of our approach, which applies operations on elements specified by violations of the selected element.

Multiple local violations $(\#6, 7, 10, 11)$*:* Finding quick fixes in these cases is possible but takes a considerable amount of time, especially if more than one solution is generated. For example, finding three solutions for case $\#7$ takes almost 3 minutes and the exploration of more than 40000 states. However, we found that even with complex DSMLs such as BPMN visiting one state only takes between $2ms$ and $4ms$, independently of the number of states explored before (at least in the scope of the measurements this held).

First solution $(\#6, 7)$*:* We found that often the quick fix generation finds a solution early on even for large models and multiple local violations, but then the majority of runtime is spent looking for alternative solutions.

To summarize, it is feasible to generate quick fixes for DSMLs, in most cases without interrupting the editing process. Our approach finds alternative solutions for local violations without considerable deviation between executions and the memory usage remains in the acceptable range (between *30MB* and *200MB* in all cases). Although in some cases the fix generation for multiple violations is costly, the generation can be interrupted without waiting for multiple solutions, thus resulting in an almost anytime-like algorithm.

VI. Related work

The quick fix generation approach presented in our paper deals with correcting local inconsistencies in models using predefined model manipulation operations. Similar approaches are found in the areas of *model construction and syntax-directed editing* and *inconsistency handling* of models. In this section we place our approach with regards to existing works.

Model construction and syntax-directed editing: Model construction deals with creating consistent models through a series of operations. In [16] models are constructed by providing hints to the designer in the form of valid operations, which are calculated using logic constraints and reasoning algorithms, to maintain global correctness. Mazanek [20] introduced an auto-completion method for diagram editors based on hyper-edge grammars, where model construction proposals are generated for incomplete models (although not inconsistent). In [10] they extended the approach for generating correctness-preserving operations for diagram editing, by identifying irrelevant or incorrect operations. This approach uses local scopes for operations (i.e. the user selects the elements where auto-completion is desired), similarly to our approach. In [15] models are transformed to Alloy specifications and automatic completion suggestions are calculated on the derived model.

These approaches present construction methods, where models are built in a monotonously increasing way, while our approach can also remove parts of the model when inconsistency handling requires it. Furthermore, these approaches translate models into some analysis formalism for generating operations, while our method works directly on the original models. Finally, the extensibility (adding and removing constraints and operations) of these approaches is limited by using derived analysis models, while our approach supports dynamic handling of inconsistency rules and operations.

Inconsistency handling: Fixing common inconsistencies in design models (such as UML) are regarded as an important challenge and several techniques were proposed for addressing it. Egyed et al. [21] presents an inconsistency resolution method for UML models, where possible consistent models are generated based on predefined inconsistency rules. The approach is restricted to operations, which change only one element at a time, while our quick fix generation approach allows the definition of complex operations.

[22] proposes an approach for generating *repair plans* is presented for EMF-based UML models. It uses *generator functions* as operations and extends inconsistency rules with information about the causes of inconsistencies, to make their search algorithms more efficient. It supports the restriction of the maximum size of the explored state-space and fitting repairs to most recent inconsistencies similarly to our approach.

The inconsistency resolution problem is solved using automated planning in [23] without manually defining operations. Instead, it generates valid models by translating design models to logic literals and executing analysis on this derived model. While planning is similar to heuristic-driven exploration, our approach does not use derived models.

Nentwich et al. [24] define a distributed approach, where operation definitions (resolving one inconsistency at a time) are generated automatically from inconsistency rules described in a logical language and incremental consistency checking is performed directly on the design models. However, the approach handles inputs as XML documents, while our technique works directly over models.

Graph transformation is also frequently used for handling inconsistencies. In [25] inconsistency checking and operations are performed on the model in response to user interface events with actions defined with triple-graph grammar. In [11] models are checked and corrected based on modeling

guidelines defined using graph transformation rules to help users in resolving a large number of violations automatically. Mens [12] proposes critical pair analysis of inconsistency rules and operations defined as graph transformation rules for improving inconsistency management by detecting incompatible resolutions and ordering solutions.

All of these approaches are similar to our quick fix generation method in using graph transformation as a formalism for capturing inconsistency rules and operations. However, our approach uses heuristic-driven state-space exploration and is able to generate quick fixes with local scopes independently of the total number of violations in other parts of the model.

VII. Conclusion

In the paper, we elaborated a novel approach for generating quick fixes (as instantly applicable, complex corrections to consistency violations) for DSMLs. Our technique is based on a heuristics-driven state exploration algorithm for solving structural constraints directly over domain-specific models. We have assessed and justified the scalability of our implementation using real-life models provided by an industrial partner.

Our implementation (accessible as an open source add-on to VIATRA2) fulfills the challenges established in Section II as follows: (i) as quick fixes are generated in an anytime fashion, the first solution is usually quickly found and presented to the user; (ii) the heuristics-guided algorithm automatically selects the best, conservative solutions, with additional fine-tuning possible as future work; (iii) as demonstrated in the evaluation, the engine is capable of generating solutions for a given local scope as well as the entire model; (iv) our high abstraction level, declarative formalism allows both the language engineer and the end user to build extensible inconsistency rules and operations at runtime.

Regarding future research, we plan to investigate various heuristics (such as dependencies between rules) as well as different solver algorithms, to extend the scalability even further. A very promising idea is to add the ability to *learn* user-selected quick fixes to a runtime knowledge base, and reuse such knowledge for consequent solution generation. This could be investigated in a model versioning and conflict management case study. Technologically, we aim to develop our current tool further to support the generation of quick fixes over generic EMF models, to enable the integration of this technology to all existing Eclipse-based modeling tools.

Acknowledgment

This work was partially supported by the SECURECHANGE (ICT-FET-231101) and CERTIMOT (ERC_HU_09-1-2010-0003) projects and the Janos Bolyai Scholarship. We would like to thank the Bank of the Northeast of Brazil (Banco do Nordeste – BNB) for providing the case study.

References

[1] Object Management Group, "Business Process Model and Notation (BPMN) Version 1.2," http://www.omg.org/spec/BPMN/1.2/.

[2] The Eclipse Project, "Eclipse Modeling Framework Project," http://www.eclipse.org/emf.

[3] J.-P. Tolvanen and M. Rossi, "MetaEdit+: defining and using domain-specific modeling languages and code generators," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ser. OOPSLA '03. New York, NY, USA: ACM, 2003, pp. 92–93.

[4] Object Management Group, "Object Constraint Language (OCL)," http://www.omg.org/spec/OCL/.

[5] SOA Tools Platform, "Eclipse BPMN Modeler," http://www.eclipse.org/bpmn/.

[6] The Eclipse Project, "EMF Validation Framework," http://www.eclipse.org/modeling/emf/?project=validation.

[7] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds., *Handbook on Graph Grammars and Computing by Graph Transformation*. World Scientific, 1999, vol. 2: Applications, Languages and Tools.

[8] F. Orejas, H. Ehrig, and U. Prange, "A logic of graph constraints," in *Fundamental Approaches to Software Engineering (FASE)*, 2008, pp. 179–198, LNCS 4961, Springer.

[9] G. Bergmann, A. Ökrös, I. Ráth, D. Varró, and G. Varró, "Incremental pattern matching in the viatra model transformation system," in *Proceedings of the Third International Workshop on Graph and model transformations*. ACM, 2008, pp. 25–32.

[10] S. Mazanek and M. Minas, "Generating correctness-preserving editing operations for diagram editors," in *Proc. of the 8th Int. Workshop on Graph Transformation and Visual Modeling Techniques. ECEASST*, vol. 18, 2009.

[11] C. Amelunxen, E. Legros, A. Schürr, and I. Stürmer, "Checking and enforcement of modeling guidelines with graph transformations," in *Applications of Graph Transformations with Industrial Relevance*, 2008, pp. 313–328, LNCS 5088, Springer.

[12] T. Mens, R. Van Der Straeten, and M. D'Hondt, "Detecting and resolving model inconsistencies using transformation dependency analysis," in *Proc. of Model Driven Engineering Languages and Systems*, 2006, pp. 200–214, LNCS 4199, Springer.

[13] Á. Horváth and D. Varró, "Dynamic constraint satisfaction problems over models," *Software and Systems Modeling*, 11/2010 2010.

[14] Á. Hegedüs and D. Varró, "Guided state space exploration using back-annotation of occurrence vectors," in *Proceedings of the Fourth International Workshop on Petri Nets and Graph Transformation*, 2010.

[15] S. Sen, B. Baudry, and H. Vangheluwe, "Towards domain-specific model editors with automatic model completion," *Simulation*, pp. 109–126, 2010, 86(2).

[16] M. Janota, V. Kuzina, and A. Wasowski, "Model construction with external constraints: An interactive journey from semantics to syntax," in *Proceedings of the 11th Int. Conf. on Model Driven Engineering Languages and Systems*, 2008, pp. 431–445, LNCS 5301, Springer.

[17] A. Balogh and D. Varró, "Advanced model transformation language constructs in the VIATRA2 framework," in *ACM Symp. on Applied Computing (SAC 2006)*. Dijon, France: ACM Press, 2006, p. 1280–1287.

[18] G. Bergmann, Á. Horváth, I. Ráth, and D. Varró, "Experimental assessment of combining pattern matching strategies with VIATRA2," *Journal of Software Tools in Technology Transfer*, 2009.

[19] P. Gilbert, "The next decade of BPM," 2010, keynote at the 8th International Conference on Business Process Management.

[20] S. Mazanek, S. Maier, and M. Minas, "Auto-completion for diagram editors based on graph grammars," in *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008.* IEEE, 2008.

[21] A. Egyed, E. Letier, and A. Finkelstein, "Generating and evaluating choices for fixing inconsistencies in uml design models," in *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, 2008, pp. 99 –108.

[22] M. Almeida da Silva, A. Mougenot, X. Blanc, and R. Bendraou, "Towards automated inconsistency handling in design models," in *Advanced Information Systems Engineering*, B. Pernici, Ed., 2010, pp. 348–362, LNCS 6051, Springer.

[23] J. Pinna Puissant, T. Mens, and R. Van Der Straeten, "Resolving Model Inconsistencies with Automated Planning," in *Proceedings of the 3rd Workshop on Living with Inconsistencies in Software Development*. CEUR Workshop Proceedings, 2010, pp. 8–14.

[24] C. Nentwich, W. Emmerich, and A. Finkelstein, "Consistency management with repair actions," in *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, pp. 455 – 464.

[25] E. Guerra and J. de Lara, "Event-driven grammars: Towards the integration of meta-modelling and graph transformation," in *Graph Transformations*, 2004, pp. 215–218, LNCS 3256, Springer.