Model-driven architectural risk analysis using architectural and contextualised attack patterns

Shamal Faily
University of Oxford
Oxford, UK
shamal.faily@cs.ox.ac.uk

John Lyle
University of Oxford
Oxford, UK
john.lyle@cs.ox.ac.uk

Cornelius Namiluko
University of Oxford
Oxford, UK
cornelius.namiluko@cs.ox.ac.uk

Andrea Atzeni
Politecnico di Torino
Torino, Italy
andrea.atzeni@polito.it

Cesare Cameroni
Politecnico di Torino
Torino, Italy
cesare.cameroni@polito.it

ABSTRACT

A secure system architecture is often based on a variety of design and security model elements. Without some way of evaluating the impact of these individual design elements in the face of possible attacks, design flaws may weaken a software architecture. This paper illustrates how architectural and contextualised attack patterns can be used to formalise the elements of architectural attacks and possible defences. We illustrate how these patterns, and tool-support building upon them, can be used to automate an architectural risk analysis process. We demonstrate this approach using an example from the EU FP7 *webinos* project.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*methodologies,tools*

General Terms

Security, Design

Keywords

architectural risk analysis, patterns, CAIRIS

1. INTRODUCTION

It is often the case that the design of a software system does not start with a blank page, but with a bricolage of different model elements. For example, by re-using existing software components, we introduce design elements into our architecture. Assumptions about possible attacks might also influence architectural decision making. Even designers with a good understanding of both the problem and solution domains may not appreciate the implications of protocol selection, or the wording of requirements. Moreover, because the abstractions used by a designer don't always match those used by an attacker then flaws missed by the former may be found and exploited by the latter. Consequently, we need tools that help us assess the security consequences of bringing together different model elements.

In recent years, we developed the IRIS (Integrating Requirements and Information Security) meta-model to integrate concepts from Usability, Security, and Requirements Engineering [5] practice. The meta-model, which is described in detail in [3], is centered around a number of concepts which are common in each of these areas and was devised to help structure and manage usability and security design activities for different system contexts of use. This work has contributed to the development of the open-source Computer Aided Integration of Requirements and Information Security (CAIRIS) requirements management tool [4], which has been validated using real-world case studies, e.g. [6]. While the IRIS meta-model can deal with risks in the broader socio-technical environment within which a software system is situated, two further augmentations are needed for undertaking a more detailed analysis of software architectures. First, although the IRIS meta-model provides substantial support for modelling the problem domain, solution domain concepts are limited to the notion of assets. While modelling assets is necessary for architectural risk analysis, it is not sufficient as many features of an architecture warrant analysis in their own right; these include the architecture's *attack surface* – the measure of its exposure to attack – and the properties of connections between its elements. Second, model representations are needed for specifying the elements of software architecture and the attacks these need to resist. These representations need to match the thinking that designers might have about different perspectives of a system, and it should be possible for them to quickly evaluate the consequences that attack and defence elements might have on each other.

In this paper, we illustrate how modest extensions to the IRIS meta-model, together with complementary tool-support, can be used to automate an architectural risk analysis. In Section 2 we introduce meta-models for architectural patterns and contextualised attack patterns; these formalise the elements necessary to facilitate an architectural risk analysis. We show how these elements are applied in practice in Section 3. We discuss related work that complements this approach in Section 4, before concluding in Section 5.

MDSec'12, September 30 2012, Innsbruck, Austria

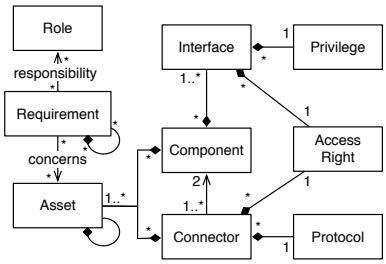

Figure 1: Architectural patterns meta-model

2. APPROACH

The aim of an architectural risk analysis is to identify design-level flaws in a software architecture. This process was first described by McGraw [14], and motivated by the claim that design flaws account for a significant number of security problems; such flaws cannot be identified by code-inspection alone. An architectural risk analysis shares many of the characteristics of classic risk analysis: it emphasises tangible assets of business value and, as a process, is knowledge intensive, requiring knowledge of both the problem domain and security expertise about potential flaws and attacks. In other respects, however, architectural risk analysis is more challenging. It relies on additional knowledge about solution-based models that form the basis of a software architecture, along with a sense of the requirements and constraints implicitly assumed when these are adopted.

The approach prescribed by McGraw for carrying out an architectural risk analysis involves carrying out three steps. In the first step, an *attack resistance analysis* is carried out to identify general flaws from the literature and knowledge bases of known attacks and, based on these, identifying potential risks and their viability. In the second step an *ambiguity analysis* is carried out to discover new risks resulting from ambiguity and inconsistency in a design. In the final step, a *weakness analysis* identifies weaknesses that might arise due to the impact of the architecture's dependencies. Interested readers may wish to refer to [14] for a more detailed presentation of this process, although Khan et al. [11] provide a more recent illustrative example based on an analysis of the Chromium browser.

Our approach involves carrying out a *model-driven* architectural risk analysis. The process for carrying out this analysis, which is based on that proposed by McGraw, is centred around the creation and analysis of two model-based constructs: architectural patterns and contextualised attack patterns. Once these models are created, these are amenable to automated analysis to determine whether a software architecture addresses known attacks, and whether any aspect of the architecture requires further investigation. The following sections describe how these constructs are defined and used.

2.1 Architectural patterns

Architectural patterns were proposed by Buschmann et al. [2] to express a model of a software system, provide a set of pre-defined sub-systems, specify responsibilities, and include rules and guidelines for organising the relationships between model elements. To capture the elements of an architectural design pattern, we introduce new concepts to the IRIS meta-model, while making use of existing concepts and relationships. Collectively, the architectural patterns meta-model in Figure 1 provides three conceptual different views of an architectural pattern.

The first of these is a component and connector view, which is expressed using UML component diagrams. This captures the runtime attributes of a system in terms of its computational elements (components) and the interaction pathways between them. Components are attached to connectors via interfaces; these are services or methods through which component interaction takes place. Interfaces are associated with an access right to indicate the level of authorisation needed to use the interface. Interfaces also have a particular privilege level. Connectors, like components, are characterised by their access rights and also by the protocol upon which the connector runs. These meta-model components are closely aligned with the model of software architecture described by Gennari & Garlan, which was recently adapted to capture the elements of a software architecture's attack surface [8]. This involves specifying the model elements associated with components and connectors, and assigning numeric privilege, access right, and protocol values to these elements. These values range between 0 and 10 and represent an element's exposure to attack; the higher the value, the greater the exposure. These values make it possible to formally evaluate the damage potential associated with interfaces, data transmitted through a connector, and untrusted data items with respect to restrictions placed on the data they contain. This formal model is described in more detail in [8].

The second is a goal view, and is characterised by the system requirements that motivate or constrain the architectural components; within the IRIS meta-model, these system requirements are expressed using KAOS (Knowledge Acquisition in automated Specification) goal trees [19]. KAOS responsibility links describe the roles responsible for satisfying the behaviour associated with requirements, while concern links are used to describe instances where requirements reference or constrain assets.

The third view is based on a module view of assets. These assets are salient concepts of value that are specific to components or connectors; this view is expressed using UML class diagrams.

2.2 Contextualised attack patterns

Attack patterns are descriptions of common methods for exploiting software that both provide an attacker's perspective, together with guidance towards mitigating them [17]. In recent years, work on attack patterns has been popularised by the development of open-source intelligence Common Attack Pattern Enumeration and Classification (CAPEC) [17] and Common Weakness Enumeration (CWE) [18] repositories. Although these repositories offer a wealth of useful attack data, the patterns are deliberately abstract in order that they can be applicable in as many contexts as possible. To provide this context and re-use as much existing model data as possible when applying these patterns, we have developed a meta-model for *contextualised attack patterns*. These model both the attack and design elements necessary to instantiate an attack within a specific context of use.

The model upon which contextualised attack patterns are

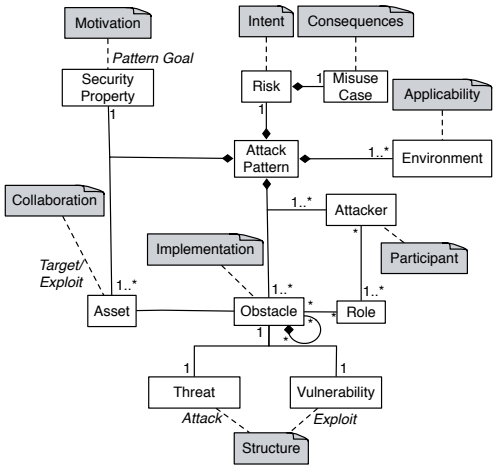

Figure 2: Contextualised Attack Patterns meta-model

based is the *Gang of Four* design pattern template [7]. However, the meta-model is based exclusively on existing concepts and associations in the IRIS meta-model. As such, when a contextualised attack pattern is introduced into CAIRIS, it introduces a new risk, together with the IRIS model elements that act as its rationale. The relationship between the contextualised attack pattern structure and the meta-model is illustrated in Figure 2 by the grey UML comment nodes; these nodes denote the name of the *Gang of Four* pattern elements.

The intent and consequences elements are used to describe the overall intent of the attack, and the external impact of the attack being successful. This impact is broader than the impact of a particular architectural pattern and is described using terminology that all system stakeholders can understand. The applicability element states the environment within which the attack pattern will be introduced. This is also the same environment within which architectural patterns will be situated to determine whether the design elements are resistant to this or other attack patterns within the environment.

The structure element describes the details of the attack itself. Attacks and Exploits are drawn from both CAPEC and CWE. The structure is closely complemented by the participant element, which models information about the attack; this includes the attacker's capabilities and motives for carrying out the attack. This information is derived from personas that have been created for possible attackers. Personas are specifications of archetypical user behaviour that are grounded in empirical data collected from representative users [15]. These add a substantial amount of context to the analysis because not only do these add a human face to the attackers behind attack patterns, these are grounded in data sources about real attackers.

To describe how the attack defined by the structure might be implemented, leaf obstacles are associated with threats and vulnerabilities and a KAOS obstacle model is defined to describe how these might arise. Like the KAOS goal model in the architectural pattern, these obstacles might concern assets, and responsibility associations describe the roles responsible for satisfying obstacles.

As Letier has observed [20], a KAOS obstacle model can be seen as a goal-driven form of a fault tree. However, unlike fault trees, our approach to obstacle modelling is closely tied to other artifacts such as previous knowledge about attacks and information about the attackers that might carry these out. Collectively, where useful statistical data about possible attacks exists, this information can help us predict the likelihood of particular obstacles being satisfied. When a probability value is specified for this likelihood then a rationale statement also needs to be provided to justify it. This is necessary because, when attack patterns are imported into a CAIRIS model, it may not be immediately obvious that the obstacle or the obstacle model arose from them. By proving this justification, we have some way of understanding the thinking that motivated this value. Based on these values, we can evaluate the probability of a particular cut of an obstacle tree based on the same equations used to evaluate the faults in a fault tree. For example, for an obstacle O_x with non mutually exclusive leaf goals O_1 and O_2, the probability of O_x (i.e. $P(O_x)$) where O_1 and O_2 are AND-refinements is $O_1 \times O_2$; where O_1 and O_2 are OR-refinements then $P(O_x)$ is $O_1 + O_2$.

Like classic design patterns, the collaboration concept describes the classes necessary to achieve the designer's intent. However, in the case of attack patterns, the classes are assets and the designers are attackers. As such, the collaborating assets are those which are targeted by threats or exploited by vulnerabilities. Closely aligned with this concept are motivating security properties of interest to an attacker realising this pattern.

2.3 Architectural Risk Analysis

We have modified CAIRIS to support the aforementioned changes to the IRIS meta-model. We have also modelled architectural patterns and contextualised attack patterns as XML Data Type Descriptions; these facilitate the import of both types of artifact directly into CAIRIS. As our approach is contingent on access to knowledge bases about possible attacks and exploits, we have also developed scripts which import information from CAPEC and CWE as directories of potential threats and vulnerabilities respectively. More information about CAIRIS's facilities for importing threat and vulnerability directories can be found in [3].

Before an architectural risk analysis can be carried out, pre-existing security, usability, and requirements models need to be created in CAIRIS. These include information about known system assets, the type of roles supported by the system's personas, along with supporting artifacts such as use cases and requirements. At this stage, it is also useful to elicit architectural attack surface meta-data for protocols, privileges, and access rights, and agree numeric values for each type.

The architectural risk analysis process itself is iterative, and entails the following four steps:

- *Architectural pattern specification*: For a specific area of architectural significance, an architectural pattern is created to encompass the component and connector, requirement, and asset views associated with this area. As McGraw suggests [14], this is the most intellectually demanding part of the process because the information necessary to populate the pattern needs

to be elicited from various sources, including design documentation and source code. It is also necessary to involve other designers and domain experts to validate the architectural pattern as it is specified. For this reason, although this is first step in the process, the pattern itself will invariably be revised throughout the process.

- *Attack resistance analysis*: In this step, we begin to populate the contextualised attack pattern template based on potential security concerns that may be associated with the pattern. This includes searching the imported knowledge bases for the pattern structure elements, and identifying attacker personas with the ability to carry out the identified attack. If the existing attacker personas do not have either the capabilities or motives for carrying out the attack, then it may be necessary to create a new, more meaningful attacker persona. The process for doing this is beyond the scope of this paper, but is described in more detail by [1].

- *Ambiguity analysis*: this involves eliciting potential causes of the attack using the obstacle models in the contextualised attack pattern. The threat and vulnerability elements act as initial leaf obstacles and, by considering the attack from the perspective of the attack persona, the leaf obstacles are abstracted to identify why these occur. As further obstacles are elicited, these are refined to identify other potential threats and vulnerabilities. As this model evolves then, where possible, probability values are assigned to obstacles, and potential goal and responsibility links are assigned to known system assets and roles referenced in the contextualised attack pattern.

- *Weakness analysis*: to understand the impact that the environment has on the architectural pattern, we import the contextualised attack pattern into CAIRIS, and introduce the previously created architectural pattern into the same environment as the attack pattern. In addition to automatically generating diagrams such as those shown in Figures 3 and 4, the damage potential across the interfaces, channels, and untrusted surfaces associated with architectural pattern are also calculated. Where assets are associated with both the architectural and contextualised attack pattern, potential threats and vulnerabilities to the architectural pattern are highlighted, and — where concern and responsibility links are common to both requirements and obstacles — potential obstacles that obstruct architectural patterns are highlighted. As the number of potential obstacles might be large, these are ordered by probability, where the most likely obstacles are listed first.

After the final step, any identified threats and vulnerabilities which are adequately treated by architectural pattern requirements are marked by the designer, together with how effective the treating requirement is. Similarly, goals which are obstructed by candidate obstacles are also marked. At this point, we repeat the process to refine the architectural pattern based on new information about components, connectors, assets, or requirements.

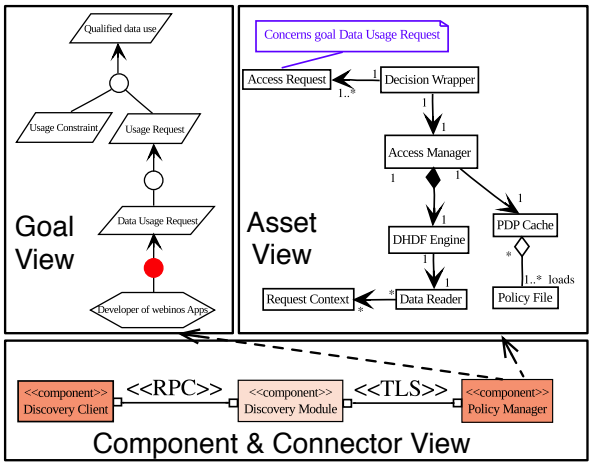

Figure 3: Asset, Goal, and Component and Connector View of Policy Manager Architectural Pattern

3. EXAMPLE

To demonstrate our approach, we show how it can be used to support the architectural risk analysis of the policy framework for the EU FP 7 *webinos* project. *webinos* is a federated software platform for running web applications consistently and securely across mobile, PC, home media, and in-car systems. More information about the project is described in [22].

3.1 Architectural pattern

The *Policy Manager* architectural pattern illustrated in Figure 3 specifies the policy framework developed for *webinos*. The policy framework is summarised in [13], and it is from this that the asset view was derived. To develop the requirements view, it was necessary to review other sources of information such as prototype code for the policy management software itself.

Following subsequent validation of this pattern with the authors of [13], a component and connector view was devised to illustrate how other *webinos* components might interact with the Policy Manager component. In the component and connector view in Figure 3, a software application (Discovery Client) on a device may wish to discover other similar applications running on other devices. However, before *webinos*' discovery capabilities can be invoked, the Discovery Module component needs to establish if the application is authorised to access the requisite resources.

Once the architectural pattern was imported into CAIRIS, the attack surface metrics were automatically calculated and used to colour the component nodes in the component and connector view. Shades of red are used to determine the comparative size of the exploitable attack surface, where components coloured with a darker shade have a larger surface than those with a lighter shade.

3.2 Attack Resistance Analysis

For this example, we assumed that test APIs had been unintentionally introduced into the *webinos* platform. Based on keywords associated with this reason, we search the CAPEC

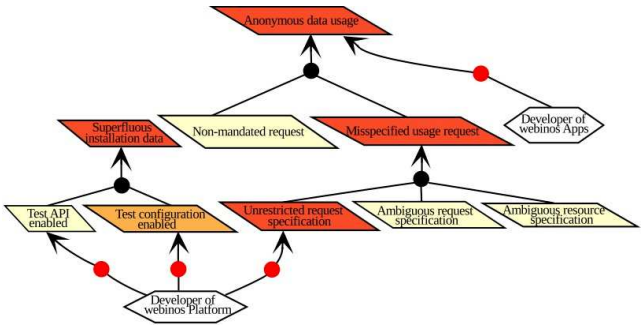

Figure 4: Obstacle model derived from CAPEC-121 and CWE-770

and CWE knowledge bases imported into CAIRIS and identified a potential attack, i.e. CAPEC-121: Locate and Exploit Test APIs; this attack takes advantage of the associated weakness CWE-770: Allocation of resources without limits or throttling. Based on the associated information in CAPEC and CWE, we determined that one of the *webinos* attack personas — Ethan — has both the capability and motives for carrying out this attack. It was also determined that the threat would target application data by taking advantage of possible weaknesses that might arise in the access request itself. More information about Ethan and other *webinos* attacker personas can be found in [21].

3.3 Ambiguity Analysis

Figure 4 shows the obstacle model developed on the basis of the initial leaf obstacles *Test API enabled* (arising from CAPEC-121) and *Unrestricted request specification* (arising from CWE-770). Reflecting on why both these obstacles might have occurred led to the elicitation of obstacles that gave rise to them. For the first obstacle, this was the inclusion of superfluous installation data as part of a *webinos* platform release; this included not only test APIs, but also test configuration data. In the case of the second obstacle, this arose as a result of a usage request for a policy resource being misspecified. The model indicates that this misspecification might also arise because of ambiguity in the request, or ambiguity about the policy resource itself. However, the probability of each of these leaf goals is not the same. No probability was set for the other leaf obstacles of *Misspecified usage request*, however, based on the implementation logic for handling request specifications, there is a surprisingly high probability (70%) that the *Unrestricted request specification* obstacle might be satisfied. Because the leaf obstacles to *Misspecified usage request* are OR-refinements (as denoted by the filled black circle), then its probability is the sum of its leaf goals which, in this case, is also 70%.

As the figure also shows, the obstacles are coloured in shades of red based on their probability; the higher the probability of an obstacle occurring, the darker the shade.

The process of carrying out the ambiguity analysis also gave more clarity about what Ethan's goals might be in carrying out this attack based on possible cuts of the obstacle model. As the obstacle model shows, misspecified usage requests might arise because a *webinos* application developer anonymised (perhaps unintentionally) data usage requests.

3.4 Weakness Analysis

Because the access request asset is part of the asset view and is also targeted by the attack pattern then, when the architectural pattern is imported, CAIRIS asks for a requirement which treats this attack.

CAIRIS also identifies that the data usage request requirement in the architectural pattern shares the same responsibility link and concern links as the anonymous data usage obstacle in the attack pattern. As the requirement states that users shall specify how the data being requested will be used, and the obstacle is satisfied when users do not specify how they will use the data they are requesting, then the obstacle is a reasonable obstruction to the goal.

There is no obvious requirement associated in the architectural pattern to address either the attack or the obstacle. For this reason, it is necessary to iterate the process in order to refine the architectural pattern.

4. RELATED WORK

The approach taken by IRIS for addressing risks is similar to that adopted by CORAS: a model-driven approach for risk analysis [12]. Both approaches are grounded in a meta-model encompassing the elements necessary to perform a risk analysis, and both are supplemented by a software tool. However, IRIS has a weaker focus on risk modelling; its meta-model is much broader and primarily concerned with the concepts necessary to support interactive secure system design; this includes information about possible attackers and various other elements of a system's context of use. Moreover, because its scope is much broader, IRIS does not prescribe any particular method; instead, it relies on a process framework that practitioners can instantiate based on the characteristics within which IRIS will be used; this process framework is described in more detail by [3].

The approach we take for eliciting exploitable assumptions relies on KAOS and, in particular, obstacle models. Work by Heyman et al. [9] propose an alternative approach for eliciting such assumptions by formally specifying requirements using Alloy [10], together with the assumptions these rely on. These security requirements act as assertions that reference an architectural model of components that is similarly modelled in Alloy. Our conceptual model for architectural patterns is a super-set of the architectural meta-model proposed by Heyman et al., suggesting the efficacy of their approach for providing additional analysis of architectural patterns. Heyman et al. also imply further alignment with our work by drawing on the usefulness of KAOS obstacle models for discovering analysable assumptions.

Our approach for incorporating probabilities into contextualised attack pattern obstacles is inspired by recent work by Sabetzadeh et al. [16]. In comparison, our approach is less sophisticated in that it says little about how probability values are independently validated beyond providing individual rationale statements. However, our approach does benefit from the close relationship between obstacles and other model elements; these provide additional evidence for making claims about individual probability values. Moreover, our approach does not necessarily preclude a more sophisticated strategy, such as that prescribed by Sabetzadeh and his colleagues.

5. CONCLUSION

This paper has presented an approach for carrying out a model-driven architectural risk analysis. In doing so, we have made several contributions. First, we have shown how, with the aid of pre-existing requirements and usability models, and only modest extensions to the IRIS meta-model, it is possible to specify the elements of a software architecture. By doing so, we can also automate some of the drudgery associated with analysing an architecture's attack surface, and identifying obstacles that conflict with architectural requirements. Second, we have shown that by leveraging pre-existing model data and the IRIS meta-model, it is possible to create contextualised attack patterns that can be used to evaluate potential weaknesses in a software architecture. Third, we have shown that building and applying these patterns aligns with McGraw's architectural risk analysis process, as demonstrated by an illustrative example.

Based on architectural patterns we have created to encapsulate different parts of the *webinos* software architecture, we have recently applied this approach to explore how resistant *webinos* is to different types of known web application attack, as well as particular security concerns raised by the project team. The results of this study will be described in future work.

6. ACKNOWLEDGEMENTS

The research described in this paper was funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2).

7. REFERENCES

[1] A. Atzeni, C. Cameroni, S. Faily, J. Lyle, and I. Fléchais. Here's Johnny: a Methodology for Developing Attacker Personas. In *Proceedings of the 6th International Conference on Availability, Reliability and Security*, pages 722–727, 2011.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture: a system of patterns*. Wiley, 1996.

[3] S. Faily. *A framework for usable and secure system design*. PhD thesis, University of Oxford, 2011.

[4] S. Faily. CAIRIS web site. `http://github.com/failys/CAIRIS`, July 2012.

[5] S. Faily and I. Fléchais. A Meta-Model for Usable Secure Requirements Engineering. In *Proceedings of the 6th International Workshop on Software Engineering for Secure Systems*, pages 126–135. IEEE Computer Society, 2010.

[6] S. Faily and I. Fléchais. User-centered information security policy development in a post-stuxnet world. In *Proceedings of the 6th International Conference on Availability, Reliability and Security*, pages 716–721, 2011.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[8] J. Gennari and D. Garlan. Measuring attack surface in software architecture. Technical Report CMU-ISR-11-121, Carnegie Mellon University, 2012.

[9] T. Heyman, R. Scandariato, and W. Joosen. Security in context: Analysis and refinement of software architectures. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, pages 161 –170, july 2010.

[10] D. Jackson. *Software abstractions: logic, language and analysis*. MIT Press, Cambridge, Mass., 2006.

[11] M. Khan, M. Munib, U. Manzoor, and S. Nefti. Analyzing risks at architectural level. In *Information Society (i-Society), 2011 International Conference on*, pages 231 –236, 2011.

[12] M. S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer, 2010.

[13] J. Lyle, S. Monteleone, S. Faily, D. Patti, and F. Ricciato. Cross-plaform access control for mobile web applications. In *Policies for Distributed Systems and Networks (POLICY), 2012 IEEE International Symposium on*, pages 37–44, 2012.

[14] G. McGraw. *Software Security: Building Security In*. Addison-Wesley, 2006.

[15] J. Pruitt and T. Adlin. *The persona lifecycle: keeping people in mind throughout product design*. Elsevier, 2006.

[16] M. Sabetzadeh, D. Falessi, L. Briand, S. Alesio, D. McGeorge, V. Ahjem, and J. Borg. Combining goal models, expert elicitation, and probabilistic simulation for qualification of new technology. In *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, pages 63 –72, nov. 2011.

[17] The MITRE Corporation. Common Attack Pattern Enumeration and Classification (CAPEC) web site. `http://capec.mitre.org`, July 2012.

[18] The MITRE Corporation. Common Weakness Enumeration (CWE) web site. `http://cwe.mitre.org`, July 2012.

[19] A. van Lamsweerde. *Requirements Engineering: from system goals to UML models to software specifications*. John Wiley & Sons, 2009.

[20] A. Van Lamsweerde and E. Letier. Integrating obstacles in goal-driven requirements engineering. pages 53–62, Apr 1998.

[21] webinos Consortium. User expectations on privacy and security. `http://webinos.org`, February 2011.

[22] webinos Consortium. webinos web site. Available from `http://webinos.org`, March 2012.