Model-based development of security requirements

Eduardo B. Fernandez
Florida Atlantic University, Dept. of Computer and Electrical Eng. and Computer Science
Boca Raton, FL, USA
ed@cse.fau.edu

and

Sergio Mujica
Universidad Santo Tomas, Escuela de Informatica
Santiago, Chile
sergio.mujica@gmail.com

Abstract
We present a model-based approach using two dimensions to propagate security restrictions: along the lifecycle and along the architectural levels. We apply security patterns to perform this propagation. We believe that this double propagation can be very effective for security and reliability. This approach can also facilitate the security analysis of the system and can be used to verify compliance with regulations. We have developed a methodology to apply these ideas and we are extending it to make it more powerful, in particular to increase its level of security and to add to it also reliability concerns. The extensions include two new metamodels for security requirements and a validation approach.

Keywords: security patterns, security requirements, reliability, conceptual models, compliance, model-driven design

1. INTRODUCTION

Many current systems have serious security problems. We believe that a good way to have secure systems is to build applications and systems software in a systematic way, where security is an integral part of the lifecycle. The same applies to reliability. If we want a system which is secure and reliable, both security and reliability must be built together. If we build not only applications but also middleware and operating systems in the same way, we can build systems that not only are inherently secure, i.e., they can withstand attacks from malicious applications, but also can resist errors. All security and reliability constraints should be defined at the application level, where their semantics are understood, and propagated to the lower levels. The lower levels provide the assurance that the constraints are being followed, i.e. they guarantee that there are no ways to bypass these constraints.

It appears that the best way to provide a unified view of security in the presence of myriad implementation details of the component units is to use abstraction through models, correcting code flaws is a hopeless task for large and complex systems. In particular, we apply abstraction through the use of security and reliability patterns. Our idea is to be able to describe all architectural relationships in any type of network by means of patterns. Patterns are encapsulated solutions to recurrent system problems and define a vocabulary that concisely expresses requirements and solutions without getting prematurely into implementation details [1]. The description of architectures using patterns makes them easier to understand, provides guidelines for design and analysis, and provides a way to make their structure more secure in the presence of a growing amount of attacks, at the same time increasing their overall reliability. Most books on software architecture, e.g. [2], describe the importance of patterns but none applies them systematically to build architectures with high security and high reliability requirements. Security patterns describe mechanisms that can stop specific security threats and they also embody principles of good security design [8].

Our approach to building secure architectures involves using two dimensions to propagate security restrictions, along the lifecycle and along the architectural levels. We believe that this double propagation can be very effective for security. In the past, systems have been built where some user interactions are not possible, e.g. the Burroughs machines only allowed high-level language access to the architecture. If we can restrict users in similar although not so restrictive ways, we can significantly improve system security. This approach would also facilitate the security analysis of the system. Applying security along all the stages of the lifecycle is now an accepted good practice for security and these two aspects are synergistic. Propagation along architectural levels is our idea and requires a relatively complete catalog of patterns. Patterns defined at the application level are reflected in lower-level patterns. We have built an extensive catalog of security patterns that cover all the architectural layers [3].

To apply this approach we need an appropriate methodology, patterns are not very useful without a systematic way to apply them. We have developed such a methodology [4]. Its main ideas are that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with security principles. Another basic idea is the use of patterns to guide security at each stage. Patterns are applied to cover all architectural levels. We are extending it in the ways described in this paper, so we can see this paper as a set of extensions to make this methodology more powerful, in particular increase its level of security and add to it also reliability and compliance aspects. The extensions include two new metamodels for security and reliability requirements and a way to validate these requirements. However, the models presented here are applicable to any security methodology.

The rest of the paper is organized as follows: Section 2 summarizes our methodology. Section 3 presents the metamodels. Section 4 considers the validation of these models, while Section 5 compares our results to others. We end with some conclusions.

2. A METHODOLOGY TO BUILD SECURE AND RELIABLE SYSTEMS

This methodology considers the following development stages [4]:
Domain analysis stage: Conceptual models to cover relevant areas are defined. Legacy systems are identified and their security implications analyzed. General security or reliability constraints can be applied at this stage in the form of patterns.
Requirements stage: Use cases define the required interactions with the system. We relate threats to use cases. Considering the activities in each use case we can enumerate threats in a systematic way [5]. The security test cases for the complete system are also defined at this stage.
Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. Security patterns describe security models or mechanisms. We can build a conceptual model where repeated applications of security patterns apply the required security constraints [6].
Design stage: We express the abstract security patterns identified in the analysis stage in the design artifacts, e.g. interfaces, components, distribution, and networking.
Implementation stage: This stage requires reflecting in the code the security rules defined in the design stage. In this stage we can also select specific security packages or COTS, e.g., a firewall product, a cryptographic package.

Figure 1 summarizes the lifecycle of an application, showing the use of security patterns in each stage. We assume that the relevant domain analyses have been performed in advance, i.e we have one or more domain models from which we can take some sub-models to build the application. Reliability aspects are handled similarly and concurrently. We consider first the use cases of the system. A use case is a complete interaction of a user with the system. We enumerate systematically the possible threats to the use case activities. Use cases are also used to find security test cases for the complete system once implemented as executable code. From the use cases we build the conceptual model of the application, which describes precisely the functional requirements of the system. We superimpose on the functional aspects, security patterns to control the identified threats. These are abstract security patterns, defining basic security functions and free of implementation aspects. The conceptual model plus the security patterns define the security requirements of the application. The secure conceptual model is transformed into a design model that considers software aspects and where architectural security patterns are added based on the requirements of the analysis stage. Finally code is produced from the design model complemented with COTS components that implement security mechanisms, e.g. firewalls. Currently we know how to get to the secure conceptual model but we don't have a precise way to transform this

model into a secure design model. We are here making the early stages more precise in order to make easier the transition to a design model.

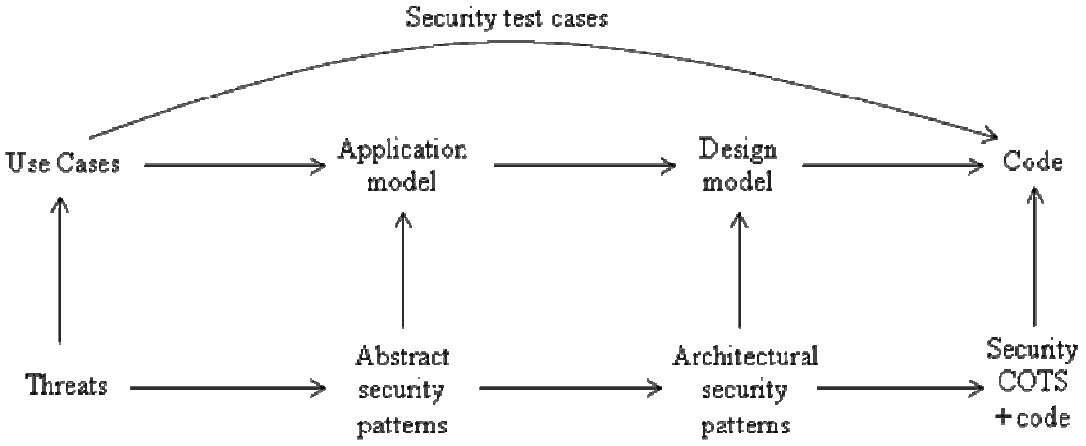

Figure 1: Patterns and lifecycle of an application.

3. METAMODELS FOR SECURITY REQUIREMENTS

Figure 2 shows a metamodel that relates domain models to application models. A **Conceptual Model** describes functional relationships between entities; it realizes a set of use cases (textual descriptions) into a more precise representation. A **Domain Model** describes functional relationships between entities related to a specific domain, e.g. finance or electrical engineering. An **Application Model** describes the functional aspects of an application and may combine parts of several domain models (takesFrom) as well as its own set of conceptual entities. Domain models and application models are conceptual models. A conceptual model may include analysis patterns and ad hoc functional units (not patterns). Some analysis patterns are Semantic Analysis Patterns (**SAP**s), which correspond to a small set of coherent use cases [7]. The **Policy-enhanced Domain Model (PEDM)** and the **Policy-enhanced Application Model (PEAM)** correspond to domain and application models where we have superimposed policies for security or reliability. In particular, the PEAM inherits policies from some domain models and may add its own policies.

Figure 3 shows a metamodel to go from threats and failures to patterns. A **Threat** can be neutralized by a **Security Policy**. Similarly, a **Failure** can be neutralized by a **Reliability Policy**. **Policies** may include also **Regulations** and **Institution Policies**. Security and reliability policies are realized by **Security** and **Reliability Patterns**, respectively. A **Policy Realization Pattern** is a pattern that realizes any type of policy and consists of a few classes and associations. Security and Reliability patterns are special cases of Policy Realization patterns.

The set of use cases in the application, UC_A, are the union of some of the use cases in the domain model, UC_{Di}, and a set of new use cases, UC_{new}. Since we relate threats to use cases, new use cases mean new threats. The metamodel of Figure 3 applies both to domain models and to application models. When applied to a domain model we enumerate threats/failures based on the use cases of the domain model; when applied to the application model its threats are the union of the threats in the domain model and the threats in the new use cases.

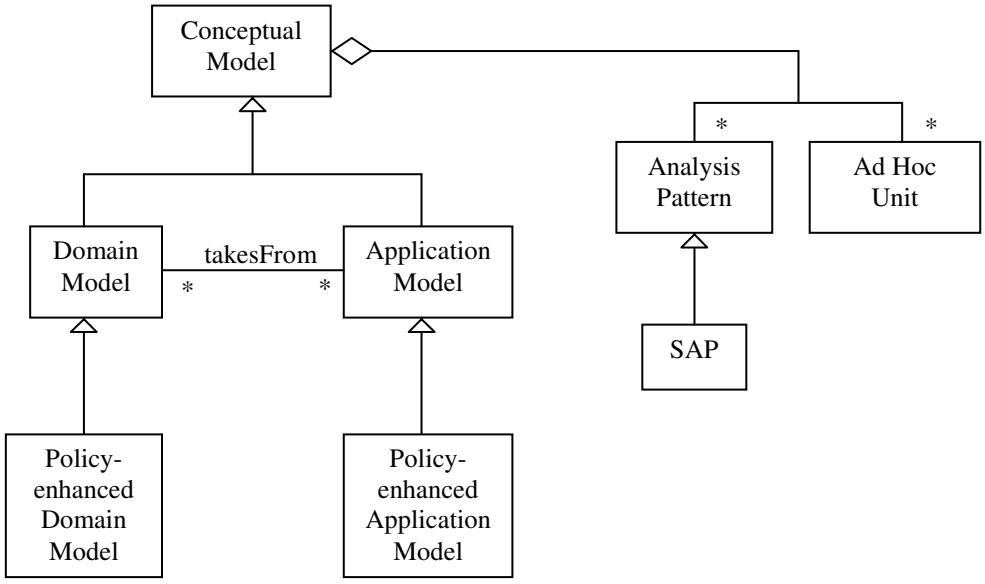

Figure 2: Relationships between domain models and application models.

To build secure applications, we start by defining first a Domain Model (DM) and superimpose on it security /reliability patterns that neutralize threats and failures and also includes all policies related to regulations and institution policies, this is the PEDM. A PEAM inherits the association 'takesFrom' and thus incorporates into it the PEDM. Since the application cannot override security/reliability constraints and regulations coming from the PEDM, it means that the effort spent developing a domain model that complies with regulations is recovered by having application models which are automatically compliant with those regulations. In previous work we developed a systematic approach to enumerate all (most?) of the threats to an application [5]. As shown in Figure 3, threats can be stopped or mitigated by appropriate policies. To these policies we must add policies that correspond to regulations and institution policies, both of which must appear in any application. All these threats must be handled by specific mechanisms in order to stop them. Our threat analysis uses the activity diagrams of the use cases of the system. We can do this analysis in the domain model for those aspects which cut across applications and in the application models for those aspects unique to the applications. We can combine this approach with analysis patterns and obtain parts of domains which incorporate all their needed security restrictions in the form of patterns, e.g. a law trial secure analysis pattern [6].

Figure 4 shows an example of these concepts. A threat of illegal data modification has been detected by some approach, e.g. [5]. Two policies acting together are able to stop this attack: Access Control to intercept all accesses and validate them; Need-to-Know to define fine granularity rules in the access control system. A third policy, Log/Audit does not stop the attack but would record the attack actions for future system improvement or prosecution. It is usually used together with the other two policies. An access control policy is realized by two patterns: Authorization, that defines a structure for access rules, and Reference Monitor, that enforces the rules [8]. We see from this figure that the four "leaf" patterns can neutralize this attack.

All the security patterns that neutralize threats cannot be defined in the DMs since some threats depend on the specific application, e.g. the goals of the attacker could be quite different in each application. However, as indicated above, we have a way to enumerate the specific threats of an application. In the DM we can add threats to parts of an application as discussed below.

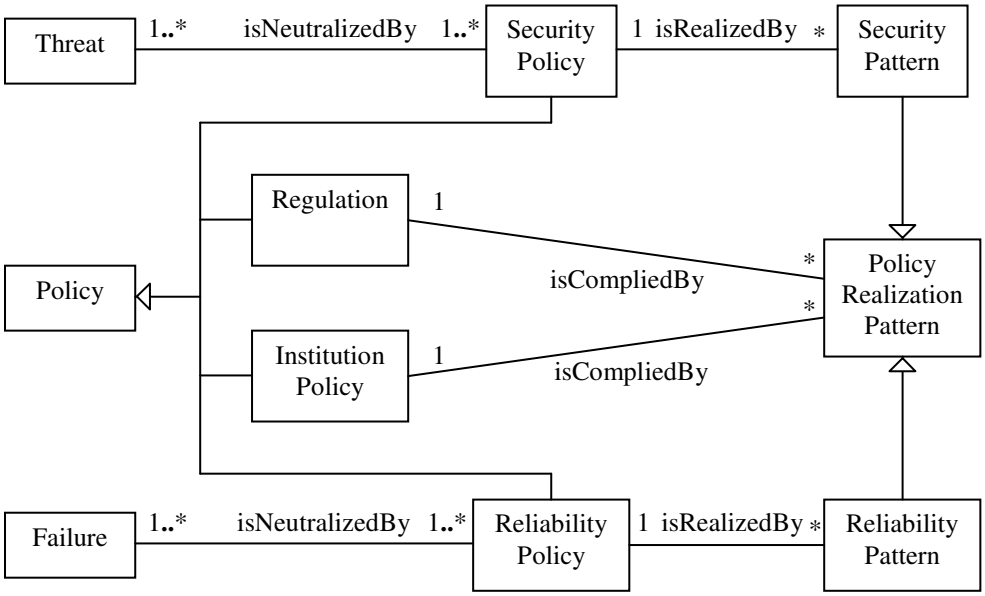

Figure 3: Metamodel for requirements and patterns

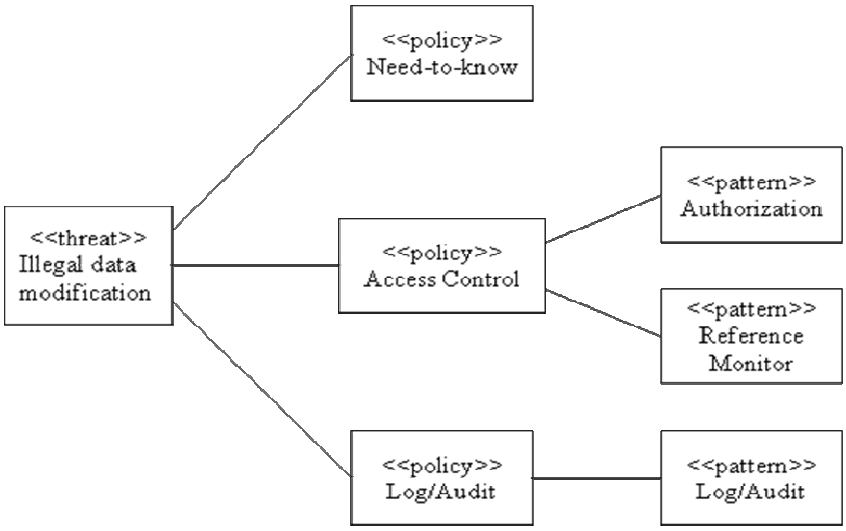

Figure 4: Example of going from threats to patterns.

4. VALIDATION OF SECURE CONCEPTUAL MODELS

Finally, we ask: Can we certify an application by using security patterns? We can show that the application includes a pattern able to stop each expected threat by a simple matching of threats/failures to patterns. Threats can be stopped through policies. If we have a pattern for each policy, we can consider the system secure at the model level, although there may still remain code vulnerabilities (see below). However, code-based attacks are limited by a good architectural design; for example, a virus compromising an email unit, cannot reach its address book if it has some access control for the address book.

Formally, if T = set of threats for a system, Py = set of policies that stop or mitigate T, and Pt = set of patterns that realize Py, we have then: T → Py → Pt, for all t in T → there exists Py → Pt that is, if for all the threats

in the system, we can find a set of policies that can be realized as a set of patterns that neutralize those threats, we consider the system to be secure.

In the use of policies we need to separate the mechanism that realizes or implements the policy from the contents of the rules or constraints defined in the mechanisms. For example, a pattern for a need-to-know policy requires a mechanism where we can define authorization rules, e.g. an Authorization pattern [8]. This policy adds the constraint that all the authorization rules must be defined in such a way that the users are given the minimum set of rights they need to perform their duties. A pattern for a need-to-know or similar policy does not define a specific mechanism but a way to define authorization rules.

In Figure 5, application AM1 uses domain models DM1 and DM2. Analysis of threats, errors,and compliance policies in these domain models results in adding to them patterns to neutralize or comply with these policies; these are PEDM1 ans PEDM2. Application AM1 inherits all these patterns and as such it can handle threats, errors, and compliance aspects coming from these two domains. For example, pattern p1 which stops threat t1, ,is inherited by PEAM1. However, since AM may have parts not inherited from any doamin model, it is exposed to additional threats, T1, and may require compliance with additional regulations, RAM1. Additional patterns handle the new requirements and now AM1 is able to consider all its security/reliability/compliance requirements. The new model is PEAM, which includes all the necessary patterns.

We can enumerate systematically all the threats and select all institution policies and regulations needed to build a Security/Reliability-enhanced Application Model. These policies lead to a set of abstract security mechanisms incorporated in the PEAM. We can validate this stage by verifying that all threats and failures have been handled and all institution policies and regulations are complied. The next stage is to transform this metamodel into one of a variety of design models. There may be several possible models depending on architectural decisions; for example, if we need authorization and we use web services for distribution we need to choose between SAML and XACML or a combination of both. In the design stage only interface-related classes need to be protected by authorization, other classes are internal and cannot be attacked; similarly we only need authentication in classes that have communication with remote classes or with user interfaces. After each architectural decision we need to analyze which of the threats identified earlier are still possible and how they are manifested in the chosen architecture.

Can we stop all attacks in this way? We are working only with models, but there can still be attacks due to flaws in the code, e.g. a buffer overflow condition in a procedure to withdraw money from an account. We can handle them in different ways:

- Carefully controlling which classes can be accessed through an interface. For those that need to be accessed, we can define their signatures very precisely, specifying appropriate parameter lengths and types.
- Even if the attacker succeeds in producing the buffer overflow condition, a good design can make it ineffective by not letting the attacker's procedure inherit high privileges, for example by making some kernel processes execute in user mode, not allowing the execution of data, and not giving children processes the rights of their parents.
- Procedures that implement operations in object-oriented designs are much simpler that corresponding procedures in a procedural approach. It is then much easier to check or even verify their code.

The first two arguments apply to any system, even built without using object-oriented approaches, while the last argument only applies to object-oriented systems. In both cases, we think we can get a good level of security by using only models. Complementing this approach with some code checking could be important for very critical systems. In this case, the amount of code checking should be easier to perform and not all code needs to be checked because of the second point above.

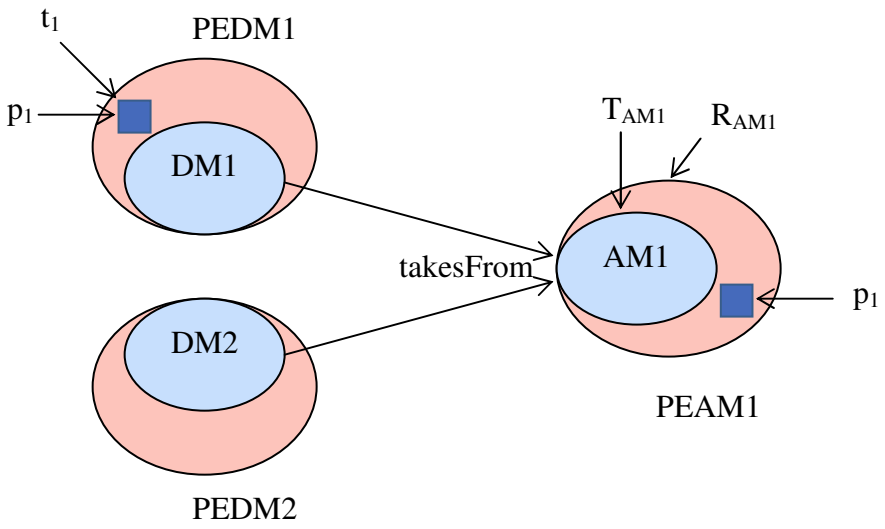

Figure 5: From domains to applications

5. RELATED WORK

Related work includes:

[9] shows an approach to validate domains but they do not try to propagate security constraints to applications. Their approach is oriented to database systems.

Mouratidis and his group use a special methodology, Tropos, to model security. Their work concentrated initially on modeling requirements. Later, they have also looked at other stages; for example how to test security along the lifecycle [10]. Instead of UML they use special diagrams and they do not use patterns. They have recently added models using UMLSec (see below). UML has the advantage of being a standard and known by most developers.

Ontologies, but no patterns, have been used in related work: [11] uses ontologies to support the development of software architectures; however, they don't consider security aspects. [12] use ontologies to develop a secure application; they use the ontologies to answer questions about the requirements but not to guide the actual design. [13] uses ontologies to model attacks and defenses to distributed systems. While ontologies add some interesting aspects, their main problem is that as each attribute is a separate entity the model explodes rapidly. Another problem, at least in these references, is that there is no architectural hierarchies and everything is at the same level, which complicates things by not allowing separation of concerns. However, they could be useful complements to UML models to answer questions about the requirements.

[14] uses aspects to model misuses and their countermeasures during secure software development. Aspects can be mapped to patterns and they can be modeled using UML. This approach skips security analysis and concentrates on the design stage.

Model-Driven Architecture (MDA) is an approach defined by the Object Management Group (OMG), which promotes the systematic use of models during a system's development lifecycle. Compared to the traditional approach in which the key artifact is the code itself, and in which the use of models is limited to documentation purposes, model-driven development aims at using models to automatically produce the final code. [15] presents an MDA-based approach to building secure systems where designers specify system models along with their security requirements and use tools to automatically generate system architectures from the models, including complete, configured access control infrastructures. Their approach includes a combination of UML-based modeling languages with a security modeling language for formalizing access control requirements. [16] presents a similar idea, of code generation from high level models, using XML. These models assume a closed

system with concerns limited first by the expressiveness of the language, and second by the available transformations.

[17] makes use of an extension of the Unified Modeling Language (UML), UMLSec, to include security-relevant information. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLSec models against security requirements. The analysis is based on model-checking specific portions of a system but apparently the approach has not been applied to a complete complex system.

The Grupo Alarcos has done a good amount of work on security requirements. [18] presents an MDA approach to develop secure business processes, including several UML models for representing security aspects of processes. [19] presents some models for selecting security requirements in software product lines. These works apply to more specialized objectives than our work and their models refer to higher-level aspects of the development cycle, specifically to business workflows.

None of these approaches considers reliability or safety aspects and only [10] uses patterns. None of them attempts to evaluate the security level reached by their approaches.

A whole set of approaches, mostly used in industry, do not use models and have to check all the application code, a daunting enterprise. This is the approach used by Microsoft, which required an enormous effort to check their operating systems. They obtained good results, increasing the quality of their systems but at a very high cost [20]. Another problem is that code changes more frequently than models.

6. CONCLUSIONS

Metamodels such as the ones presented here are useful in understanding the development process and making the development of the required artifacts more systematic. They prescribe a precise approach to build a secure/reliable conceptual model for applications. They are also useful in a Model-Driven Design approach to transform security models from one stage to the next. In our methodology, we can obtain application models where all the necessary security constraints are defined and which can be the basis of a secure design stage. Defining policies in the domain models and propagating them to applications ensures that these applications will comply with the policies, it is not up to the application builder to enforce them. This simplifies the development process and even if the application builders are not experts on security and reliability, they can produce secure and reliable applications. By enforcing security in the models we can avoid or reduce the need to check every line of code. Systems which provide secure execution environments, e.g. Caja [21], cannot control attacks that take advantage of application flaws. For approaches such as ours, it is important that the actual code follows the architectural design, i.e. we need to avoid architectural erosion [22].

Future work includes defining a precise transition to the design stage. Converting the conceptual models into software artifacts is complex because of the variety of possibilities and the need to consider all the architectural levels. We are also considering safety and availability policies in the methodology, trying to obtain models where tradeoffs between these aspects can be systematically made.

ACKNOWLEDGMENTS

The referees of the XXIX International Conference of the Chilean Computer Society and of the CLEI Journal provided useful comments.

REFERENCES

[1] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P. and Stal M. *Pattern- oriented software architecture*, Wiley 1996.

[2] Taylor, R.N., Medvidovic, N., and Dashofy, N. *Software architecture: Foundation, theory, and practice,* Wiley, 2010.

[3] Secure Systems Research Group, FAU, http://www.cse.fau.edu/~security/catalog.php

[4] E. B. Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., and VanHilst, M. "A methodology to develop secure systems using patterns", Chapter 5 in "*Integrating security and software engineering: Advances and future vision",* H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, pp. 107-126.

[5] Braz, F., Fernandez, E.B.,and VanHilst, M."Eliciting security requirements through misuse activities" *Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07).* Turin, Italy, September 1-5, 2008. pp.328-333.

[6] Fernandez, E.B. and X.Yuan, "Semantic analysis patterns and secure semantic analysis patterns", in revision for the *Int. Journal of Information and Computer Security (IJICS).* Inderscience Publishers, 2010.

[7] Fernandez, E.B.,and Yuan, X. "Semantic analysis patterns", *Procs. of 19th Int. Conf. on Conceptual Modeling, ER2000*, pp. 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf

[8] Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., and Sommerlad, P. *Security Patterns: Integrating security and systems engineering",* J. Wiley 2006.

[9] Reinhartz-Berger, I. and Sturm, A., "Utilizing domain models for application design and validation", *Inf. and Software Technology*, vol. 51, No 8, 2009, pp. 1275-1289.

[10] Mouratidis H., and Giorgini, P., "Analysing security in information systems", *Procs. of the 2nd Int. Workshop on Security in Information Systems at ICEIS 2004*, Porto, Portugal, 2004.

[11] Akerman, A. and Tyree, J. "Using ontology to support development of software architectures", *IBM Sys. Journal*, vol. 45, N0 4, 2006, pp. 813-825.

[12] Dritsas,S., Gymnopoulos, L., Karyda, M., Balopoulos, T., Kokolakis, S., Lambridounakis, C., and Gritzalis, S. "Employing ontologies for the development of security critical applications", *Procs, of the IFIP I3E Conf.*, Oct. 2005, pp.187-201.

[13] Voroviev, A. and Bekmamedova, N., "An ontology-driven approach applied to information security:", *J. of Research and Practice in Information Tech.,* vol. 42, No 1, February 2010, pp.61-76.

[14] Georg, G., Ray, I., Anastasakis, K., Bordbar, B., Toahchoodie, M., and Houmb, S.H. "An aspect-oriented methodology for designing secure applications", *Inf. and Soft. Technology*, vol, 51, No 5 (2009), pp.846-864.

[15] Basin, D.A., Doser, J., and Lodderstedt, T., "Model driven security: From UML models to access control infrastructures", *ACM Trans. on Software Engineering and Methodology*, vol. 15, No 1, (2006), pp. 39-91

[16] Nagaratnam, N., Nadalin, A., Hondo, M., McIntosh, M., and Austel, P. "Business-driven application security: from modeling to managing secure applications", *IBM Systems Journal*, vol. 44, No 4 (2005), pp.847-867

[17] Jurjens, J., *Secure Systems Development with UML,* Springer-Verlag, 2004.

[18] Rodríguez, A., Fernández-Medina, E., Trujillo, J. and Piattini, M. "Secure business process model specification through a UML 2.0 activity diagram profile", *Decision Support Systems*, vol. 51, 2011, pp.446-465.

[19] Mellado, D., Fernandez-Medina, E., and Piattini, M., "Security requirements variability for software product lines", *Procs. of the Third Int. Conf. on Availability, Reliability, and Security (ARES 2008),* pp. 1413-1420.

[20] Lipner, S. and Howard, M. "The Trustworthy Computing Security Development Lifecycle", MSDN Library, March 2005, http://msdn.microsoft.com/en-us/library/ms995349.aspx

[21] Google-Caja, http://code.google.com/p/google-caja

[22] Zhang, L., Sun, Y., Song, H., Chauvel, F., and Mei, H., "Detecting architecture erosion by design decision of architectural pattern", *Procs. of the 23rd Int. Conf. on Software Eng. and Knowledge Eng. (SEKE 2011*).