From domain models to secure and compliant applications

Eduardo B. Fernandez[1]
Departamento de Informatica, Universidad Tecnica Federico Santa Maria, Valparaiso, Chile
edfernan@inf.utfsm.cl

Sergio Mujica
Escuela de Ingeniería, Universidad Finis Terrae, Santiago, Chile
smujica@uft.cl

ABSTRACT

1. INTRODUCTION

The variety of standards and regulations (technical, institutional, government) and the increasing complexity of systems complicates the work of developers, who need to build applications which, in addition to their functional specifications, must also satisfy Non-Functional Requirements (NFRs) such as security, as well as fit platform standards. A variety of approaches have been tried [Uzu12] but most of them emphasize the application lifecycle stages starting from requirements. We believe that application building should start earlier, by first factoring out the effect of standards and regulations and by considering the effect of expected attacks in the conceptual models. For this purpose, we propose here to start from *Domain Models* (DMs), followed by the derivation of *Reference Architectures* (RA) of different types to build specialized RAs from which we can build applications which comply with specific standards and regulations and which are secured with respect to expected threats. A DM is a knowledge model, while an RA is a generic software architecture, with no platform dependencies. In our approach we consider RAs to be strictly functional, without including any NFRs, and we name explicitly RAs which incorporate some quality factor, e.g. a *Security Reference Architecture (SRA)* is an RA that includes security defenses.

We believe 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 *patterns* from which we build DMs, RAs, and SRAs. Patterns are encapsulated solutions to recurrent system problems and define a vocabulary that concisely expresses requirements and solutions without getting prematurely into implementation details [Bus96, Gam94]. *Security patterns* describe generic mechanisms that can stop specific security threats and embody principles of good security design [Fer13]. A RA is reusable, extendable, and configurable; that is, it is a kind of pattern for whole architectures and it can be instantiated into a specific software architecture by adding implementation-oriented aspects [Avg03, Mul09]. The description of architectures using patterns makes them easier to understand, provides guidelines for design and analysis, and if we also use security patterns it provides a way to make their structure more secure in the presence of a growing amount of attacks. Most books on software architecture, e.g. [Tay10], describe the importance of patterns but none applies them systematically to build architectures with high security requirements. As indicated above, applications not only must be secure but also must comply with institution and government regulations. Regulations can be expressed as policies and as such they can also be represented by patterns [Fer11a, Mas05].

Several approaches have tried to introduce systematic engineering approaches in the treatment of security, but they are not complete or are hard to use [Uzu12]. Secure systems need to be built in a systematic way where

[1] On leave from Florida Atlantic University, Boca Raton, FL, USA

security is an integral part of the lifecycle [Fer13, McG06]. If, when we build applications, we also consider the effect of middleware, operating systems, and networks as a whole, we can build systems that can withstand a whole spectrum of attacks coming from external or internal users. We believe that to build secure applications it is not effective to build a secure platform and then run on it some application made secure on its own, but that the application should match the type of platform where it will execute so they will be secure as a whole, considering the effect of platform threats. In addition, all security constraints should be defined at the application level, where their semantics are understood and propagated to the lower levels [Fer99]. The lower levels must provide the assurance that the constraints are being followed, i.e., they implement these constraints and enforce that there are no ways to bypass them. Following these ideas, we developed a secure systems development methodology [Fer06a], which considers all lifecycle stages and all architectural levels. We expanded its architectural aspects [Fer11b], and recently expanded it with process aspects. Based on that work, we have just completed defining a Cloud Security Reference Architecture (SRA) that applies this perspective by extending a Cloud Reference Architecture (RA) with security patterns [Fer14]. To design a secure system, we first need to understand possible threats to the system and we have proposed an approach to identify threats by considering the activities in each use case [Bra08]. That approach finds threats as goals of the attacker, which are then realized through the lower levels of the system. We also need to understand how the specific components of the architecture are compromised or used by an attacker in order to fulfill her objectives.

Our methodology started from the requirements stage but did not say much about the possible previous stages. We are now developing a methodology that, starting from DMs, derives applications which are secure and compliant with respect to the constraints in their DMs. By combining the RAs derived from DMs we can also produce applications that can execute according to the standards of specific platforms. We start from domain models until we end up with applications tailored for specific execution environments. We build domain models, reference architectures, and applications using patterns, some of which are functional (describe the functional aspects of the application), while others are security and policy patterns. This results in a unified approach for all the artifacts used in building software and facilitates the combination of architectures. We embed in the applications the defenses and policies which are common to several of them and then we complete the ad hoc parts of each application.

Many companies and researchers have produced RAs but they are intended for specific purposes, e.g., electric smart grids [Mic09], or cloud computing [IBM, nis11]. Usually these RAs are presented in descriptive form without attempting to be precise or rigorous. Additionally, there has been no attempt to combine these architectures or to use them to produce applications. Usually, the lower levels that consider busing, events, service requests, module interfaces, adapters, and similar, are well described, e.g., [Rai08], but the higher-level semantic aspects are assumed to be implicit in the minds of the designers or in "requirement" documents prepared by somebody else and outside the scope of the designer. This leads to a serious discontinuity that affects all Non-Functional Requirements (NFRs); in particular, those which need to be specified considering application semantics, which include security, safety, and reliability.

We describe our models using the Unified Modeling Language (UML), a semi-formal approach, complemented with the Object Constraint Language (OCL), a formal approach which is part of the UML standard [War03]. This paper extends the work of [Fer11a] where we introduced the idea of propagating security restrictions defined in domain models. Our contributions here include:

- A methodology to produce secure and compliant applications adapted to specific platforms derived from DMs. Still not complete.
- A way to derive SRAs starting from DMs and threat enumeration.

Section 2 presents some background and motivation for our work, while Section 3 shows how to go from DMs to RAs.and SRAs.Section 4 presents related work and discussion and we end with conclusions in Section 5.

2. BACKGROUND AND MOTIVATION

2.1 PATTERNS

An important development in software is the concept of *pattern*. A pattern is a solution to a recurrent problem in a given context presented according to a structured template. A pattern embodies the knowledge and experience of software developers and can be reused in new applications. Analysis patterns can be used to build conceptual models [Fer00, Fow97], design and architectural patterns are used to build flexible software [Bus96, Gam94], and security patterns can be used to build secure systems [Fer13, Sch06]. Carefully-thought patterns also implicitly apply good design principles. The typical solution provided by a pattern comes in the form of a class diagram complemented with some sequence diagrams and possibly activity or state diagrams plus (optionally) OCL constraints. This level of detail and precision allows designers to use them as guidelines and users to understand the effect of the mechanisms they represent. Patterns are also good for communication between designers and to evaluate and reengineer existing systems [Fer06b]. Design patterns are already widely accepted in industry; Microsoft, Siemens, Sun, and IBM, among others, have web pages and even books about them. Security patterns are starting to be accepted, with some companies producing catalogs in books and in the web, including IBM, Microsoft, Amazon, and Sun (Oracle). In particular, our patterns have appeared in some of those catalogs and have been used in some industrial projects. We just produced a book with more than 70 patterns and examples of their use [Fer13].

Patterns are applied in a model by *instantiation*, which has a different meaning than the way it is done in programming languages; pattern instantiation implies tailoring the pattern to fit the underlying application model, which may require renaming classes or variables and adding or removing classes and associations; in other words, patterns are not plug-ins.

We see the use of patterns as an important way to implicitly apply security principles even by people having little security experience. A complete catalog is an important tool to design complex systems. There is a need for secure system development methodologies to apply the patterns in designs and several approaches exist, including ours. As indicated earlier, we have proposed a development approach that applies security throughout the whole lifecycle and uses security patterns. As part of this work we have produced a variety of security patterns for all the architectural levels of the system as well as some special types of patterns [Fer13]. For building conceptual models we developed a type of pattern called *Semantic Analysis Pattern* (SAP), which implements a small set of coherent use cases [Fer00]. We can combine SAPs and security patterns to create Secure SAPs, which can be converted into models for secure designs where security constraints are defined [Fer07a]. *Abstract Security Patterns* (ASPs) describe conceptual security mechanisms that realize one or more security policies able to handle (stop or mitigate) a threat or comply with a security-related regulation or institutional policy [Fer08]. ASPs focus on semantic aspects and leave out implementation aspects. They are particularly useful to build SRAs. A *misuse pattern* describes how an attack involving a misuse of information is performed from the point of view of the attacker. It defines the environment where the attack is performed, countermeasures to stop it, and it provides forensic information in order to trace the attack once it happens [Fer07b].

2.2 REFERENCE ARCHITECTURES

A *Domain Model* (DM) is a model of an area of knowledge, e.g. electrical engineering, and has no software concepts. We can think of a DM as a compound pattern, including several simpler patterns that represent specific aspects of the domain. These patterns could be Analysis patterns or SAPs. A DM may also include a set of use cases, UC, that describe all the actor interactions with the DM, and a set of Roles, R, corresponding to its stakeholders (the actors).

A *Reference Architecture* (RA) is a generic software architecture, based on one or more domains, with no implementation aspects [Tay10]. It is reusable, extendable, and configurable; that is, it is a kind of pattern for whole architectures and it can be instantiated into a specific software architecture by adding platform aspects [Avg03]. There is no precise definition about what an RA should contain; [Avg03] describes an example and indicates what should be part of one, including a class model (maybe describing several levels), use cases, and deployment model. Other authors include other aspects [nis11, Str10]. Adding security defenses to an RA produces a SRA [Fer14, nis13].

3 FROM DMS TO SRAS

3.1 OVERVIEW

Figure 1 shows the possible threats and regulations at different architectural levels, showing the role of the domain stage. A DM is based on business knowledge and we derive from it RAs oriented to specific types of applications. We can consider the effect of threats in the DM and RA and add security patterns to stop them. For example, from a DM for finance we can derive an RA for bank accounts. A financial DM should comply with Sarbanes-Oxley regulations [SOX]. Financial activities are subject to a variety of threats such as illegal transfer of money. We can add security patterns in the DM to handle these threats. The banking RA should inherit the defense mechanisms of its DM and should comply with the regulations that affect banking activities. The threats in the DM correspond to the ultimate goals of the attacker, e.g. to steal money from accounts.

Applications are derived from the RAs which would incorporate business knowledge and threat defenses. Standards and regulations are applied to all the levels which they regulate. A financial application would produce transaction reports each month and it should follow the same regulations as the corresponding RA and have defenses against the identified threats. New threats appear in this level, specific to the functions of the application such as interception of monthly reports when sent to the customers. If the transactions include credit card transactions, the application would need to follow also the PCI-DSS regulations [PCI].

The applications are implemented in architectures that reflect the application standards and may add new standards. Finally the applications are deployed and configured in specific platforms. New threats appear in this level. Security monitoring is done in the deployed application.

3.2 FROM DMS TO SDMS

We can apply the threat enumeration approach of [Bra08] to the use cases of the DM. In this way, starting from a *Domain Model* (DM) we can derive a *Secure Domain Model* (SDM), which includes also its threats and their countermeasures described using Abstract Security Patterns. The threats in this level are attacks to the functions of its conceptual model and not to its implementation; for example, at this stage we do not know if some operations will be applied remotely. A conceptual threat example would be somebody drawing money from an account not her own, which would lead to a defense using content-dependent access control. Note that this type of security is abstract and does not depend on how the accounts in a bank are represented. A set of security patterns control the threats; in the example, a security pattern for content-dependent access control. To add the security patterns in the right place of the DM we follow again the procedure defined in [Bra08]. This is not now an automated procedure although a tool would be a significant help for the designer. We can connect the security patterns to the functional model manually or use a tool to connect the pattern associations in a semi-automatic way. As shown in Figure 2, these additions lead to a SDM. A compliant DM would be produced similarly.

As an example, the relevant parts of the SDM of the financial institution are shown in Figure 3, where we use UML packages and classes (a package is a named set of classes). This SDM is obtained as indicated above. The SDM includes all the classes of the DM plus the classes of the security patterns. The security functions are realized by security patterns embedded in the class model: an Authenticator [Fer13], connected to Account to validate remote user access, a Security Logger/Auditor to record transaction activity [Fer13], etc. Note that nothing is assumed about the implementation of these controls. Before computers existed, accounts were kept in paper records and the Authenticator was a bank guard who would let a customer access his record after seeing some credential. A transaction was performed by a bank clerk who would give the customer a receipt for it (this form of access control is still used).

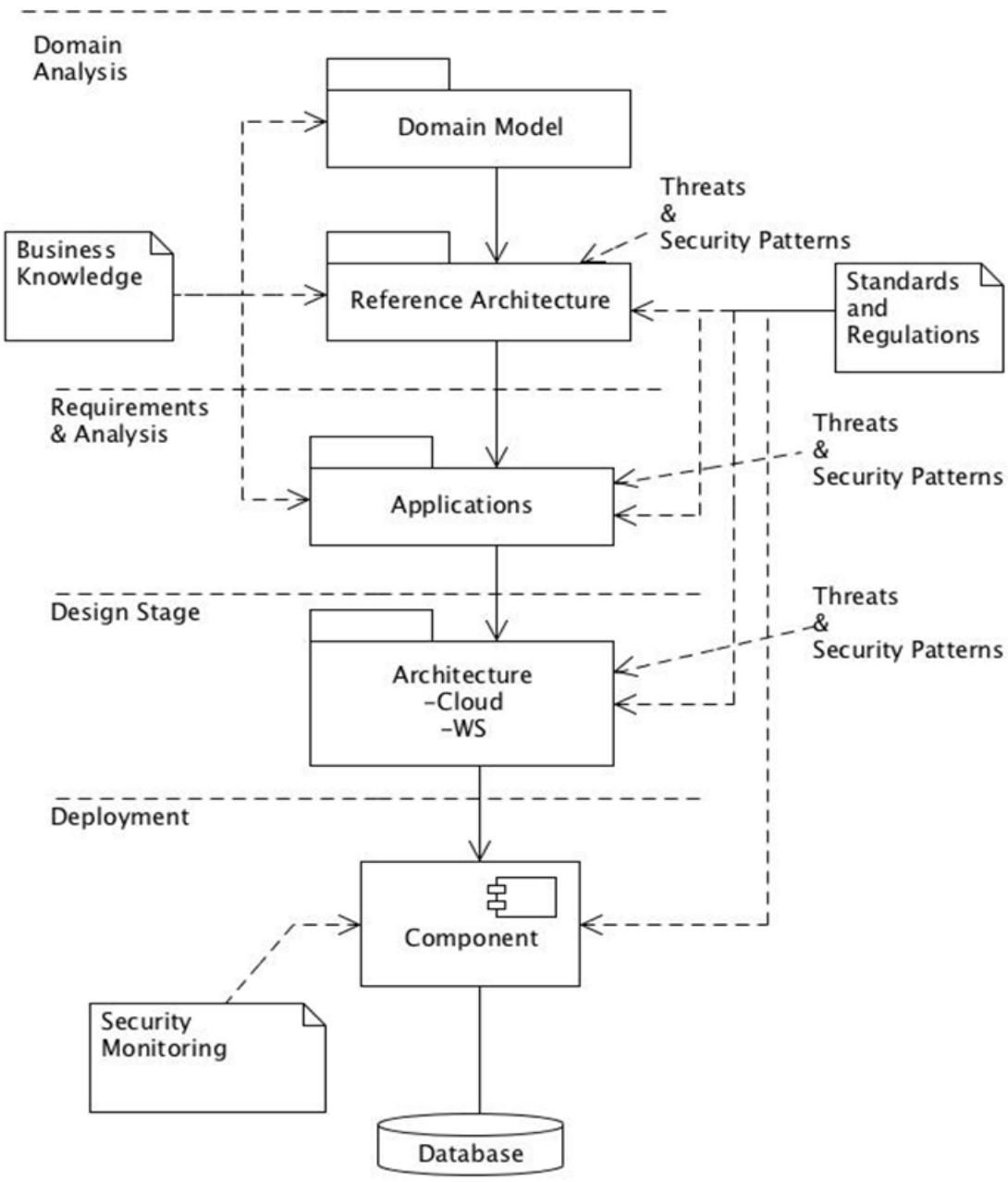

Figure 1: Threats, standards, and levels

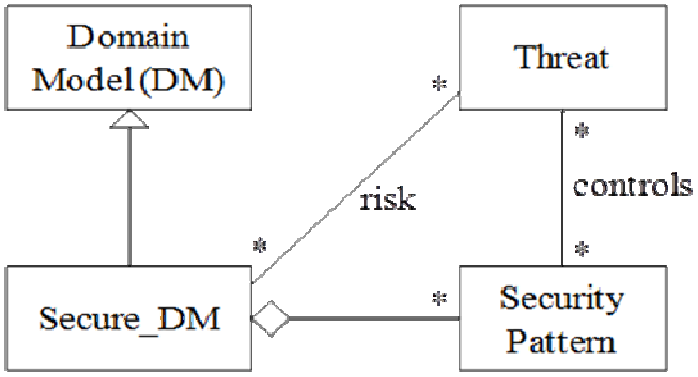

Figure 2: Producing a secure domain model

3.3 FROM SDMS TO SRAS

Similarly to DMs, we can think of an RA as a compound pattern, including several simpler patterns that represent specific aspects of the architecture. RAs include ASPs in that they are still conceptual solutions but now in the context of software architectures and with no platform dependencies. An RA may include parts of several domain models and is concerned with some type of software application. For example, a BankingRA would draw from a FinancialDM, and would include an Account pattern [Fer02], and a Party pattern [Fow97], among others (A Party pattern describes the fact that a customer can be an individual or an institution). Not all details of the FinancialDM are relevant to the BankingRA, and the BankingRA needs other concepts not in this DM, e.g., from a CreditDM. We can describe these dependencies as associations. In UML terms, the RA package would import some portions of the DM packages, including as well their use cases.

Security patterns are added to specific parts of the RA according to its threat enumeration. As indicated, a SRA includes threats and countermeasures realized as security patterns. In a Banking SRA there would be threats related to credit or accounts as well as a Content-dependent Authorization pattern to control access of customers to only their own accounts. Some of these threats are the same as in the corresponding DM but new threats may appear because of the new functionality. The ASPs of the DMs are still used in SRAs, but we will need to add more of them to control the new threats. For example, an RA attack could happen through an unprotected interface. Since we are now dealing with software architectures, we can describe misuses of information as misuse patterns, which show how the components of the architecture are used to perform the attack. However, ASPs now describe specific software mechanisms to stop attacks. Note that $UC_{RA} >= UC_{DM}$, $R_{RA} >= R_{DM}$, and $T_{RA} >= T_{DM}$, where UC is a set of use cases, R is a set of roles, and T is a set of threats. The subscripts RA and DM denote the corresponding architectures.

The application designer needs to see now how the new threats brought upon by the specific platform affect the execution of the application. We should consider how each use case of the application is now affected by the new environment. In this case, the final evaluation of security must include both the application threats as well as the new threats of the platform, and both types must be properly handled.

The mapping of an application to design artifacts requires a characterization of these artifacts in a precise way. When the application is mapped to a cloud platform for example we need to match the requirements defined in the application as conceptual countermeasures to the actual level of security provided by the cloud platform described by its security architecture. We can in this way select an appropriate platform from commercial offerings. The transformation of business requirements into concrete architectures is an interesting problem in itself [Sal12], although we will not study it here.

We need to check that no application threats, e.g. an illegal transfer of money from a bank account, are enabled by the new environment; for example, by giving access to accounts to some cloud provider employee or failing to control access as was specified in the application model.

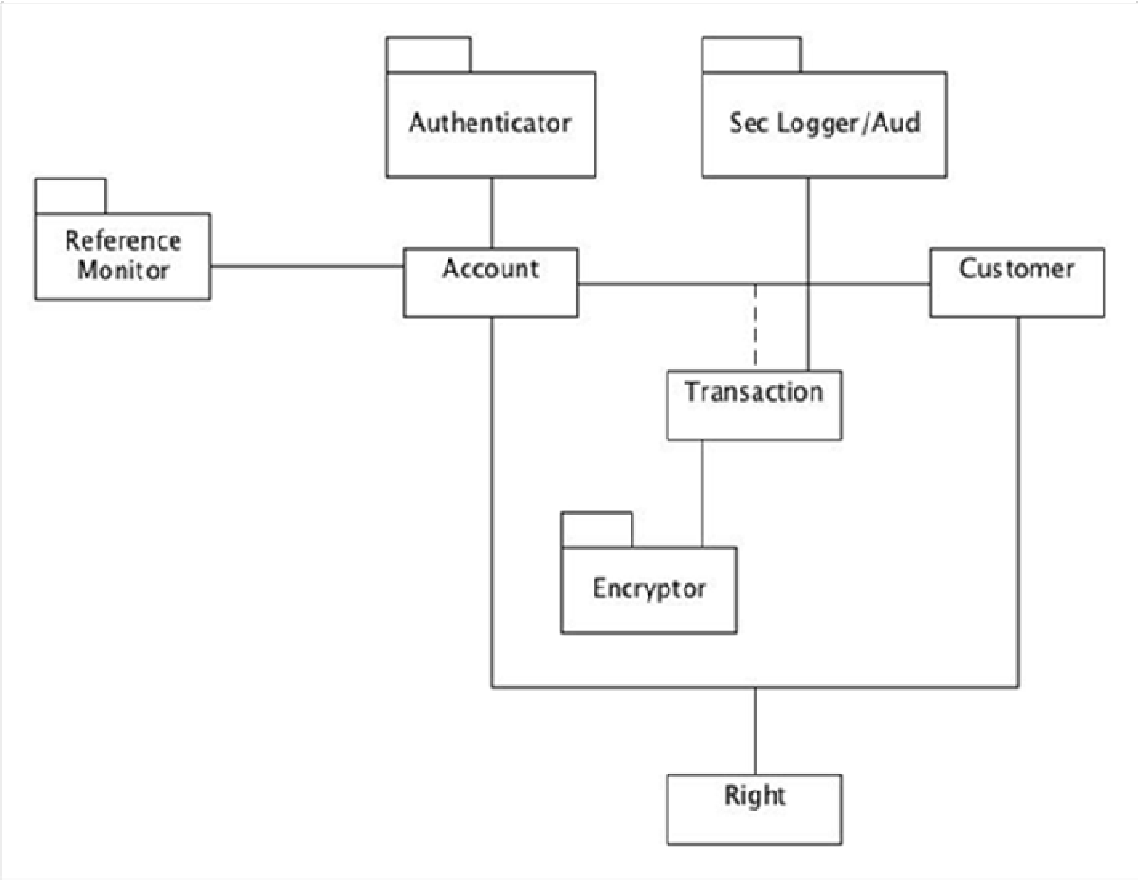

Figure 3: SDM of a financial institution

4 RELATED WORK AND DISCUSSION

[Rei09] shows an approach to validate domains but they do not try to propagate security constraints to applications. Their approach is oriented to database systems. We have not found any work about general production of secure and compliant applications. There are a few works about generating compliant or secure applications for a specific environment. [Rai08] describes the generation of AUTOSAR-compliant applications, while [Bel01] describes the generation of Java agents compliant with the FIPA standard. Neither of these two approaches consider security or can adapt to different platforms. There is also some relation to Model-Driven Architecture (MDA) [Ken02], and to Software Product Lines but they concentrate in producing one type of specific application, not a set of applications with different constraints.

We can skip or stop using any of the steps in the generation of applications; i.e., we could only use some DM, or a specific RA, or a SRA, or any combination thereof. After we stop, we can use Model-Driven Engineering (MDE) approaches [Ken02], or continue using more traditional methods. For example, we can produce a banking application from a SRA, which contains the semantics of financial applications plus the use cases, threats, and defenses of a Bank_RA. From then on, we can find further threats and defenses and we can transform its conceptual model into a design model. Adding a set of security patterns to a DM or an RA does not produce conflicts or inconsistencies because we are adding services to a functional model; each instantiation of a security pattern adds security services in specific parts of the functional model. Deriving RAs from models and other RAs from existing RAs does not produce semantic inconsistencies because we are just doing class particularization, we add more classes or attributes in subclasses, as is commonly done in object-oriented design. A clear advantage of our approach is easy traceability, we can see from where a particular rule or pattern has been derived by searching the models used in generating an application.

Applications are derived from the RAs, from which they inherit business knowledge, defenses, and regulations. A financial application would produce transaction reports each month, it should follow the same regulations as the corresponding RA, and should have defenses against the identified threats of the RA. New threats appear in this level, specific to the functions of the application such as interception of monthly reports when sent to the customers. If the transactions include credit card transactions, the application would need to follow also the PCI-DSS regulations [PCI]. The applications are implemented in architectures that need to reflect the application regulations and may add new regulations of their own. Finally, the applications are deployed and configured in specific platforms. New threats appear in this level. Security monitoring may be done in the deployed application following the structure defined by the RAs.

5 CONCLUSIONS

We have shown a systematic approach to produce secure and compliant applications. Of course, not all the threats and regulations can be applied or derived from domain models and RAs; we still need to consider the specific aspects of each application by applying a methodology such as [Fer06a] to find the patterns necessary to stop the new threats and comply with specialized regulations. However, a good part of the work is already done and we can be sure that these applications have defenses against the threats of their DMs and their RAs as well as being compliant with regulations that apply to their DMs and RAs.

Future work includes: Going from SRAs to compliant applications by combining architectures. Another aspect is the analysis of how DM and RA evolution affect existing applications. Evolution of a DM affects its derived RAs and we need to find a way to synchronize changes [Sal12]. Another direction is building a tool to generate applications in a convenient way for application developers who are not experts on security. Combinations with MDE approaches such as [Dia11], appear feasible because we only need to add our RAs to their transformation models.

REFERENCES

[Avg03] P. Avgeriou, "Describing, instantiating and evaluating a reference architecture: A case study", *Enterprise Architecture Journal*, June 2003.

[Bel01] F. Bellifemine, A. Poggi,and G. Rimassa, "JADE: A FIPA2000 compliant agent development environment", *Procs. of the Fifth international conference on Autonomous agents (AGENTS'01)*, ACM, 216-217.

[Bra08] F. Braz, E. B. Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" *Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'08).* In conjunction with the *4th International Conference on Trust, Privacy & Security in Digital Business (TrustBus'08)*, Turin, Italy, September 1-5, 2008. 328-333.

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns,* Volume 1, Wiley, 1996.

[Dia11] S. Diaw, R. Lbath, and B. Coulette, Specification and implementation of SPEM4MDE, a metamodel for MDE software processes", *Proc. International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, Miami – USA, 646-653.

[Fer99] E. B. Fernandez, "Coordination of security levels for Internet architectures,"
Proceedings of the 10th Intl. Workshop on Database and Expert Systems Applications, 1999, 837-
841 http://www.cse.fau.edu/~ed/Coordinationsecurity4.pdf

[Fer00] E. B. Fernandez and X. Yuan, "Semantic analysis patterns", *Proceedings of the 19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195.

[Fer02] E. B. Fernandez and Y. Liu, "The Account Analysis Pattern", *Procs. of EuroPLoP (Pattern Languages of Programs).*

[Fer06a] E. B. Fernandez, M. M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "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, 107-126.

[Fer06b] E. B. Fernandez and N. Delessy, ""Using patterns to understand and compare web services security products and standards", *Proceedings of the Int. Conference on Web Applications and Services (ICIW'06)*, Guadeloupe, February 2006. IEEE Comp. Society, 2006.

Fer07a] E. B. Fernandez and X. Y. Yuan, "Securing analysis patterns", *Procs. of the 45th ACM Southeast Conference (ACMSE 2007)*, March 2007, Winston-Salem, North Carolina.

[Fer07b] E. B. Fernandez, J. C. Pelaez, and M. M. Larrondo-Petrie, "Attack patterns: A new forensic and design tool", *Procs. of the Third Annual IFIP WG 11.9 Int. Conf. on Digital Forensics*, Orlando, FL, Jan. 29-31, 2007. Chapter 24 in *Advances in Digital Forensics III*, P. Craiger and S. Shenoi (Eds.), Springer/IFIP, 2007, 345-357.

[Fer08] E. B. Fernandez, H. Washizaki, and N. Yoshioka, "Abstract security patterns", Position paper in *Procs. of the 2nd Workshop on Software Patterns and Quality (SPAQu'08)*, in conjuction with the *15th Conf. on Pattern Languages of Programs (PLoP 2008)*, October 18-20, Nashville, TN.

[Fer11a] E. B. Fernandez and Sergio Mujica, "Model-based development of security requirements", *CLEI (Latin-American Center for Informatics Studies) Journal.* Vol. 14, No 3, paper 2, December 2011 Special issue of best papers presented at SCCC 2010, Antofagasta, Chile.

[Fer11b] E. B. Fernandez, Nobukazu Yoshioka, Hironori Washizaki, and Michael VanHilst, "An approach to model-based development of secure and reliable systems", *Procs. Sixth International Conference on Availability, Reliability and Security (ARES 2011)*, August 22-26, Vienna, Austria.

[Fer13] E. B. Fernandez, Security patterns in practice—*Designing secure architectures using software patterns*, Wiley 2013 Series on Software Design Patterns, June 2013.

[Fer14] E. B. Fernandez, R. Monge and K. Hashizume, "A Security Reference Architecture", submitted for publication.

[Fow97] M. Fowler, *Analysis patterns – Reusable object models*, Addison-Wesley, 1997.

[Gam94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, Mass., 1994.

[HIP] Health Insurance Portability and Accountability Act, http://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Accountability_Act

[Ken02] S. Kent, "Model driven engineering", *Procs. of IFM 2002*, 286-298.

[Mas05] Fabio Massacci, Marco Prest, and Nicola Zannone, Using a Security Requirements Engineering Methodology in Practice: the compliance with the Italian Data Protection Legislation". *Computer Standards & Interfaces*, 2005, v. 27, no 5, 445-455.

[McG06] G. McGraw, Software security: *Building security in*, Addison-Wesley 2006.

[Mic09] Microsoft Power and Utilities, *Smart Energy Reference Architecture*, Oct. 2009.

[Mul09] G. Muller and P. van de Laar, "Researching reference architectures and their relationships with frameworks, methods, techniques, and tools", *Procs. 7th Ann. Conf. on Systems Eng. research (CSER 2009)*.

[nis11] Cloud Computing Reference Architecture." 2011. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505

[nis13] Cloud Computing Security Reference Architecture, http://collaborate.nist.gov/twiki-cloud-Computing/pub/CloudComputing/CloudSecurity/NIST_Security_Reference_Architecture_01.16.2013-clean.pdf

[PCI] Cisco and other companies, "PCI-Compliant Cloud Reference Architecture", August 2011.

[Rai08] D. Rai, T. Jestin, and L. Vitkin, "Model-Based Development of AUTOSAR-Compliant Applications: Exterior Lights Module Case Study," SAE *Int. J. Passeng. Cars - Electron. Electr. Syst.* 1(1):84-91, 2009, doi:10.4271/2008-01-0221.

[Rei09] I. Reinhartz-Berger, and A.Sturm, "Utilizing domain models for application design and validation", *Inf. and Software Technology,* vol. 51, No 8, 2009, pp. 1275-1289.

[Sal12] M. Salnitri, F. Dalpiaz, and P. Giorgini, "Aligning service-oriented architectures with security requirements', *Proceedings of the 20th International Conference on Cooperative Information Systems (CoopIS 2012)*. pp. 232–249. http://disi.unitn.it/~salnitri/publications/09_12_Salnitri_alignment.pdf (last retrieved November 29, 2013).

[SOX] One Hundred Seventh Congress of the United States of America, "Sarbanes-Oxley Act of 2002", January 23, 2002. http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf

[Tay10] R. N. Taylor, N. Medvidovic, and N. Dashofy. Software *architecture: Foundation, theory, and practice*, Wiley, 2010.

[Uzu12] A. V. Uzunov, E. B. Fernandez, and K. Falkner, "Engineering Security into Distributed Systems: A Survey of Methodologies", *Journal of Universal Computer Science*, Vol. 18, No. 20, pp. 2920-3006.

[War03] J. Warmer and A. Kleppe, *The Object Constraint Language* (2nd Ed.), Addison-Wesley, 2003.

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.