Patterns Combining Reliability and Security

Ingrid A. Buckley, Eduardo B. Fernandez, and Maria M. Larrondo-Petrie
Dept. of Computer & Electrical Engineering and Computer Science
Florida Atlantic University
Boca Raton, USA
ibuckley@fau.edu, ed@cse.fau.edu, petrie@fau.edu

***Abstract*—We are developing a methodology to combine security and reliability. One aspect of this fusion implies developing patterns that combine both objectives. The Secure Reliability (SecRel) and Reliable Security (RelSec) are hybrid patterns that combine security and reliability. The SecRel pattern applies security to control the functions of a reliable system, while the RelSec pattern applies reliability to the functions that provide security. We show how these patterns relate to our methodology, and in which architectural levels they could be used.**

Keywords-*software lifecycle; software patterns; reliability; reliability patterns; security; security patterns.*

I. INTRODUCTION

Reliability is a key system characteristic that is an increasing concern for current systems. Greater reliability is necessary due to the new ways in which services are delivered to the public. Services are used by many industries, including health care, government, telecommunications, tools, and products. The lack of reliability in many systems has encouraged research efforts to find ways to improve this situation. Applications have become very complex and their reliability is a current concern.

Typically, reliability is provided through redundancy, checking and monitoring, aspects which are usually added after a system is built. A good amount of work, e.g. [11, 12, 14], has been done to include reliability in systems. There is also a large amount of work on security patterns [17]. Similarly as we did for security [8], we propose here adding reliability throughout the software development life cycle. In our approach, we start by identifying the possible failures in a system. By analyzing UML activity diagrams for all use cases and considering possible failures in each activity, we can enumerate possible service failures in applications [5]. Once failures are identified, we apply appropriate policies, realized as patterns, which will stop or mitigate these failures. In some critical parts of a system we also want to be able to provide security and reliability at the same time.

We can combine security and reliability using patterns. A pattern is an encapsulated solution to a recurrent problem in a given context. Design patterns [10] embody the experience and knowledge of many designers and when properly catalogued, they provide a repository of solutions for useful problems. Initially used for improving code, patterns are becoming more and more used to build secure and reliable systems [3, 6]. We present here two of these patterns. The Secure Reliability (SecRel) pattern applies security to a reliable system, while the Reliable Security (RelSec) pattern applies reliability to the functions that provide security in a secure system.

Section 2 discusses an approach for a secure and reliable lifecycle considering space and time aspects, including a metamodel for reliability requirements, Sections 3 and 4 present the RelSec and SecRel patterns, respectively. Section 5 provides some conclusions.

II. RELIABILITY IN SPACE AND TIME

A good way to define precise relationships between concepts in software development is to express them in metamodels. Our approach involves enumeration of failures and their origins and finds policies and patterns to handle them, so we can express their relationship as shown in Figure 1. A **Fault** manifests itself as an **Error**. If the error is not contained, it can manifest itself into a **Failure,** which indicates that the system is not following its specifications [1]. A **Policy** can avoid or handle a failure. A **Pattern** realizes the policy that can handle the fault; some examples of reliability patterns are given in [3, 4]. If we can enumerate all faults and have appropriate patterns to handle them, we can build reliable systems. We can define also patterns needed to comply with regulations.

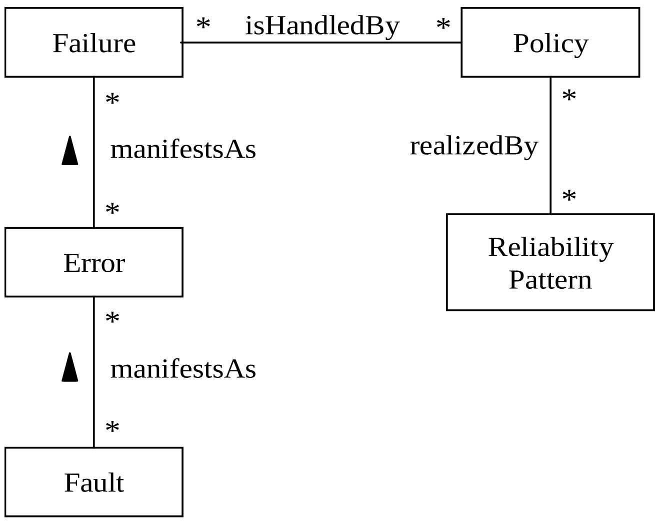

Figure 1. A metamodel for reliability concepts

The development of software applications starts from the requirements expressed normally as use cases. The use cases can be converted into a conceptual or analysis model. Analysis is a fundamental stage since the conceptual model can be shown to satisfy the requirements and becomes the skeleton on which the entire system can be built.

Reliability aspects in a computer system are expressed differently at different architectural layers (levels) and stages of the software development process. Figure 2 shows where we need to apply reliability along the stages of the system lifecycle and the architectural levels. An appropriate degree of reliability is required in each tier of the computer system layers and in each stage; also the expression of reliability varies. An important point illustrated in this diagram is the fact that the requirements must be carried over along time and along space:

1. User Interface – This is the highest level and is the user's point of contact for with the system. Usability is an important aspect for reliability and should be reflected in the interfaces.
2. Applications – The application tier of the system consists of services and programs that carry out useful operations. This tier invokes functions that are requested by the user, and should always be available when needed.
3. Middleware – Manages the interactions between the applications, DBMS, OS, and users of a system.
4. Database Management System (DBMS) – this tier is where data is stored. Failures may affect most of the applications.
5. Operating System (OS) – This level manages and synchronizes all the functions and resources within the system. Its reliability is fundamental because failures here affect all the applications
6. Data Communication – This is part of the OS and manages how information is passed throughout the system.
7. Hardware - This is the lowest level of the architecture, where instructions are executed. Its failures affect the whole system.

The requirements define the degree of reliability that the application needs. In the analysis stage, given the reliability requirements, we match the identified failures to the set of reliability mechanisms needed to stop these failures identified. If the failure is hardware based then redundancy and diversity are appropriate. If the failure is based on software, we need diversity or other approaches.

The design stage is governed by the reliability mechanisms identified in the analysis stage, which are selected to prevent failures. This may have two or more levels to describe implementation-oriented features. Web services and clouds are typical distributed architectures used in practice.

The implementation stage follows directly after the design stage; here reliability is realized in the form of code and COTS components. The whole process is iterative where some stages or parts may need to be redone. Our idea is to combine security and reliability aspects in each stage and each level. We realize reliability mechanisms by applying patterns and a catalog of them is needed to assist the architects and designers when building the system. We have produced some reliability patterns [3, 4]; here we add two patterns that combine reliability and security.

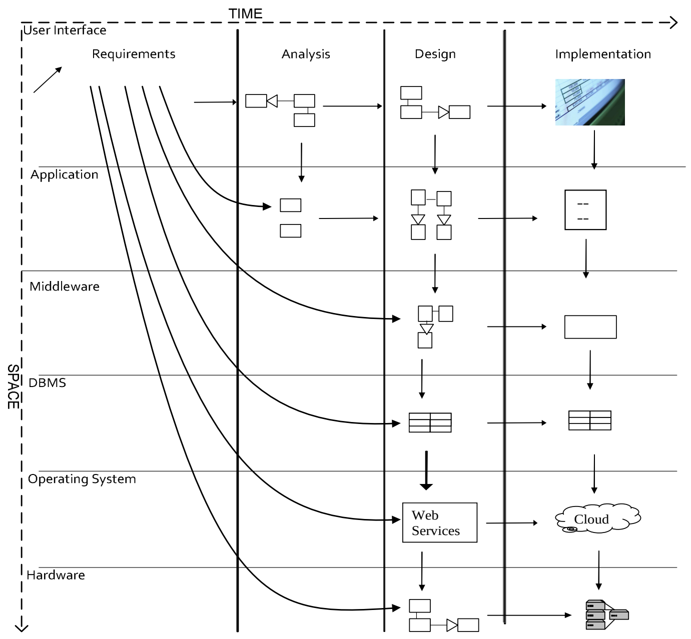

Figure 2. Reliability in Space and Time

III. THE SECURE RELIABILITY PATTERN

Intent

This pattern intends to control the use of reliable services or services that have a direct impact on system reliability. We can have services implemented using different reliability mechanisms but users may have access to only some of them. The misuse of some services may have a strong effect on system reliability.

Example

Consider a SCADA system which consists of field units, a central controller and communications networks. The field units are controlled by the central controller which is usually connected to corporate networks and or the Internet. Attackers may be able to input commands to the field units resulting in damage or disruption.

Context

Critical systems or applications that need a high degree of security and reliability to operate successfully and where we have services implemented with different degrees of reliability according to their criticality.

Problem

We can have services implemented using different reliability mechanisms but users may have access to only some of them. The misuse of some services may have a strong effect on system reliability. How do we restrict the use of the system services that affect reliability? The solution to this problem is affected by the following forces:

Reliability:

- We need to control the level of reliability of the system.
- We can have different implementations of some services according to their criticality.
- Some services affect the system reliability if improperly used; that is, security attacks can affect the reliability of the system.
- The total overhead should be reasonable.

Security:

- The system requires a given level of security. Errors can affect the security of the system.

Solution

Separate those services which could affect the reliability of the system and apply to them Role-Based Access Control [16] so that only authorized roles can use them; apply also a least-privilege policy [6]. This protects the confidentiality and integrity of the service. The structure of this pattern is illustrated in Figure 1.

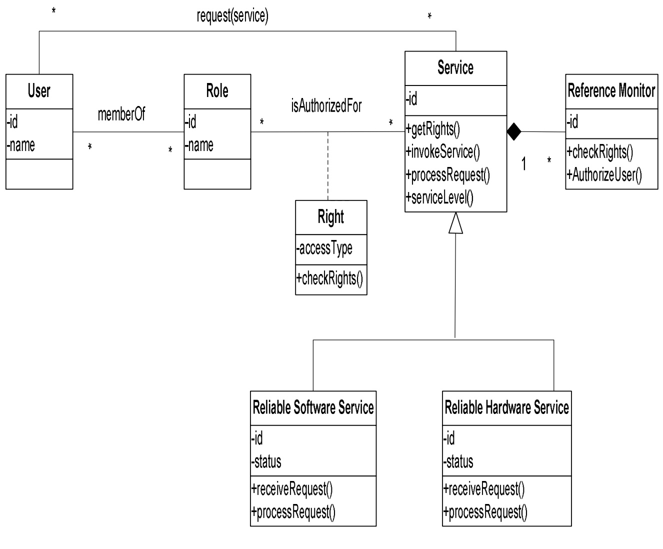

Figure 3. Class Diagram for the Secure Reliability Pattern

Structure

In Figure 3, every user is a member of a **Role**, and each Role has specific **Rights** associated with it. The **Service** entity implements a Strategy pattern [10], which chooses a **Software** or **Hardware Service,** depending on the needs of the application. The **Reference Monitor** enforces the authorized use of the service.

Dynamics

One of the dynamic aspects of the SecRel pattern is described using a sequence diagram for the use case “User sends a request for a service”, shown in Figure 4.

UC: User sends a request for a service.
Summary: A user requests a reliability-sensitive service. The request is validated by a Reference Monitor.
Actors: User
Description:

1. A user requests a service with a given level of reliability or reliability-sensitive.
2. The service invokes a Reference Monitor to check if the user is authorized.
3. If the user is authorized, the request is passed to the service. If not authorized, the request is rejected with a violation message to the user.
4. The requested service processes the request.

Post condition:
The request has been processed or rejected.

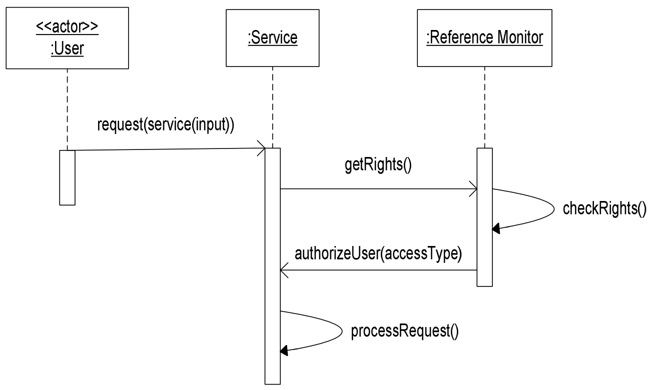

Figure 4. Sequence Diagram for the UC “User requests a service”

Implementation

To implement the Secure Reliability pattern the following is required:

1. We must have a set of reliable mechanisms or a set of reliability-sensitive services.
2. A reference monitor system is required to check the rights associated with each role before a service is used.
3. The reference monitor or an authorization service must perform authentication before a user is authorized

4. If the user is remote, a secure channel can be employed to ensure that the request of a user is passed securely so as to protect the request in transit.

Consequences

The SecRel pattern presents the following advantages:

- We can control the confidentiality and integrity of reliability-sensitive services.
- We can control the use of different degrees of service reliability by selecting different reliability mechanisms.
- The overhead of access control is small.

The pattern also has some possible liabilities:

- The authentication and authorization of users take time.
- Services may be replicated to increase their reliability; however this can increase the system overhead in maintaining them.

Known Uses

- Motorola's Canopy Platform is a wireless broadband system which enables extending broadband networks to deploy data, voice, and video applications [13]. This product uses part of the components used in the SecRelc pattern.
- Boeing's P-8 is a military derivative of the Boeing Next-Generation 737-800. It is an advanced anti-submarine and anti-surface warfare aircraft [2]. P-8 uses part of the SecRel pattern.

Related patterns

- Various reliability and fault tolerance patterns which include the Active Replication [2], TMR (Triple Modular Redundancy), and NVP (N-Version programming patterns.
- This pattern can be seen as a variation of the Role-Based Access Control pattern of [17].
- We need Authentication before we can apply Authorization [17].

IV. THE RELIABLE SECURITY PATTERN

Intent

This pattern intends to perform reliable authorization enforcement by applying reliability mechanisms to the Reference Monitor and to the Authorization rules.

Example

Consider a SCADA system which consists of field units, a central controller and communications networks. The field units are controlled by the central controller which is usually connected to corporate networks and or the Internet. Because operations can be carried out over a network, this raises a security concern. Also, the field units are usually in separate geographical locations from the central controller, therefore extreme weather or tampering can affect them. Usually it is also difficult to access them physically to repair damages, which raises a general reliability concern. If an error occurs in the security system, attackers can perform malicious actions that can disrupt the operation of the system.

Context

Critical systems or applications that need a high degree of security and reliability to operate successfully and where we have services implemented with different degrees of reliability according to their criticality.

Problem

How can we ensure that authorization is always performed correctly in the presence of errors? The solution to this problem is affected by the following forces:

Security:

- The system should always perform authorization correctly in the presence of errors. Otherwise, we will have security violations.
- The total overhead of the reliability mechanisms should be reasonable.

Reliability:

- Security services should define and enforce security constraints in a reliable way.

Solution

Apply reliability mechanisms to the Reference Monitor and to the Authorization rules.

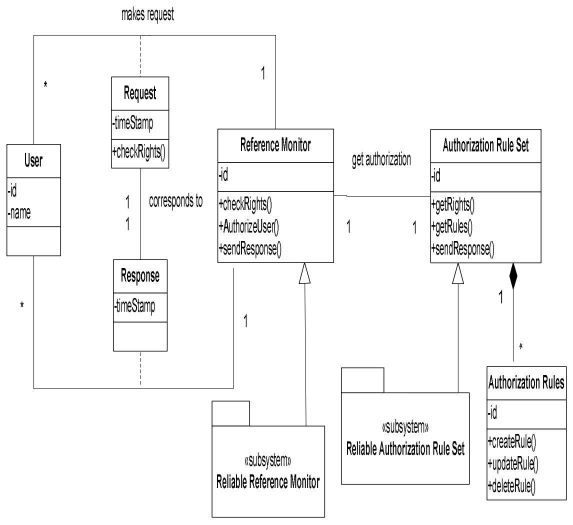

Figure 5. Class Diagram for the Reliable Security Pattern

Structure

The structure of this pattern is illustrated in Figure 5. All user **Requests** are evaluated by the **Reference Monitor,** which has access to the **Authorization Rule Set**. When the user sends a request, the **Reference Monitor** intercepts it and checks the rights (rules) associated with the request.

The **Reliable Reference Monitor** incorporates standard reliability mechanisms [3], The **Authorization Rule Set** also includes a reliability mechanism. The **Response** defines the decision from the Reference Monitor and must match the Request.

Dynamics
Figure 6 shows a sequence diagram for the use case "User requests a service".

UC: User requests a service.
Summary: A user sends a request, and the request is validated by a Reference Monitor.
Actors: User
Description:
1. The user requests some service.
2. If the user is authorized his request is processed.
3. A response is sent in response to the user's request.

Post condition:
The request has been approved or rejected.

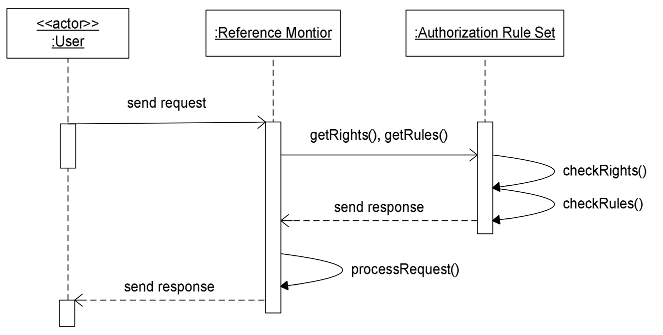

Figure 6. Sequence diagram for the UC "User requests a service"

Implementation

To implement the Reliable Security pattern the following is required:
1. We need a variety of reliability mechanisms that can be applied to the security services, e.g., those described in [3, 4].
2. The reference monitor must perform authentication before a user can send a request.
3. If the user is remote, a secure channel can be employed to ensure that the request of a user is passed securely so as to protect the request in transit.

Consequences

The RelSec pattern presents the following advantages:
- We can implement different degrees of reliability for the security services by selecting different mechanisms.
- Security checks will be reliable because all security components are reliable.
- The overhead of access control can be small because we can use reliability mechanisms that are not very complex and can be controlled to have small overhead.

The pattern also has some possible liabilities:
- The reliability mechanisms add some overhead.
- The system is more complex compared to a system without reliability mechanisms.

Known Uses

- Trumba Connect is web-hosted active event publishing solution that provides organizations with a two-way communication vehicle between events published on their websites and the personal calendaring systems used by their site visitors [19]. It uses part of the RelSec pattern.

Related patterns

- The Authentication pattern provides facilities for authenticating a user in a system [17].
- The Authorization pattern provides a way to define authorization for the users to the resources of a system [17].
- The Reference Monitor pattern checks if the process has the rights to access the object [6, 17].
- The Strategy pattern can be used to select the most suitable options to apply reliability [10].

V. DISCUSSION

The use of patterns is still not widespread in industrial institutions. Design patterns are used in large companies but many smaller companies only do coding, they don't even use models. Our work tries to encourage the use of models to build complex systems; building these systems without models will result in systems which are faulty and insecure. Models allow catching errors early in the lifecycle, which results in development savings [15]. Security patterns have reached a mature level and are starting to be used in industry for secure software development. Reliability patterns are less developed and no catalogs exist yet. Both types of patterns can help developers who are not security or reliability experts to build better systems by providing them with packaged proven solutions. Combining security and reliability in the form of patterns makes their work even easier for some applications where we need services with these two aspects. However, isolated patterns are not useful, we need pattern classification approaches and complete software development methodologies. The use of patterns fits well with Model-Driven approaches [14]; we can define reliable and secure services in domain models that can be used as starting points for critical applications [9].

VI. RELATED WORK

There is a significant amount of work on security patterns, including catalogs of patterns [17], surveys [21], and analysis of their uses and possibilities, e.g., [20]. The same is not true for reliability patterns; only a few patterns have appeared, e.g., [3, 16]. Security patterns have shown to be quite useful for designing secure systems [8] and the same can be expected for reliability patterns. Before that happens, we need a good and complete catalog of reliability patterns. As far as we know, nobody has tried to combine security and reliability patterns, although methodologies trying to combine these two aspects already exist, e.g., [14]. Some work tries to build directly reliable architectures by applying the patterns through tactics [11]. Another approach tries to insert the patterns in specific points in the architecture [18]. We believe that a systematic methodology, considering all the stages of the lifecycle is necessary. Reliability, similarly to security, must be incorporated in all the stages of the software lifecycle, adding these features in the code has shown to be ineffective. Patterns like the ones presented here combine aspects of security and reliability and can be used in systems that require a high level of security or reliability at least in some services.

VII. CONCLUSIONS

There are situations, mostly in critical systems, where the need to apply security to reliable systems and where the authorization systems should not fail. The two patterns presented here attempt to combine aspects of reliability and security to allow the implementation of systems that require very high levels of both features. The patterns are a part of our methodology to build critical systems but also have independent value.

Patterns provide a clear way for inexperienced designers to add reliability or security into their designs, but they require a good catalog of patterns that can fit all the system needs; these two patterns can be part of such a catalog. We already have a fairly complete catalog of security patterns [7, 16], so we need to find more reliability patterns as well as combined patterns as those shown here. The patterns presented here can be used in the analysis stage but they need to be extended to reflect the environment where they will be used; e.g., reliability in web services [4]. These patterns also are relevant to any level of the architecture, implemented in the appropriate technology; for example, if we determine in the use cases if a particular service is highly critical, we can add one of these patterns to the conceptual model of the application; the reliable model can then be reflected to the database or operating system levels. If web services are used for distribution, the patterns can define reliable or secure architectures for some web services.

ACKNOWLEDGMENTS

The referees provided valuable suggestions that helped improve this paper.

REFERENCES

[1] A. Avizienis, J. C. Laprie, and B. Randell, “Fundamental concepts of dependability”, UCLA CSD Report No. 010028.

[2] Boeing, “P8”, http://www.boeing.com/defense-space/military/p8/index.html, 2010. (Last accessed: August 29, 2011)

[3] I. A. Buckley and E.B. Fernandez, "Three patterns for fault tolerance" , Procs. of the OOPSLA MiniPLoP, October 26, 2009. http://www.refactory.com/miniploppapers/FTPatts.pdf. (Last accessed: August 29, 2011)

[4] I. A. Buckley, E.B. Fernandez, G. Rossi, and M. Sadjadi, "Web services reliability patterns", short paper in the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE'2009), Boston, July 1-3, 2009, pp. 4-9.

[5] I. A. Buckley and E.B. Fernandez, “Enumerating failures to build reliable systems”, submitted for publication.

[6] E. B. Fernandez, S. Mujica, and F. Valenzuela, "Two security patterns: Least Privilege and Secure Logger/Auditor.", Procs. of Asian PLoP 2011. http://patterns-wg.fuka.info.waseda.ac.jp/asianplop/program.html#papers. (last accessed August 30, 2011)

[7] E. B. Fernandez, “Patterns for operating systems access control”, Procs. of PLoP 2002, http://www.hillside.net/plop/plop2002/. (Last accessed: August 29, 2011)

[8] 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, pp. 107-126.

[9] E. B. Fernandez and S. Mujica, “Model-based development of security requirements”, accepted for the CLEI (Latin-American Center for Informatics Studies) Journal.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable object-oriented software, Boston, Mass:Addison-Wesley, 1994.

[11] N. Harrison, P. Avgeriou, and U. Zdun, “On the impact of fault tolerance tactics on architecture”, ACM SERENE 2010, London, UK, April 13-16, 2010, pp. 9-18, doi: 10.1145/1479772.1479775.

[12] M. R. Lyu, “An integrated approach to achieving high software reliability”, Procs. IEEE Aerospace Conference, Aspen, Colorado, March 21-28, 1998, vol. 4, pp. 123-136, doi: 10.1109/AERO.1998.682162.

[13] Motorola Inc, “Motorola’s Canopytm platform” http://www.ptsupply.com/pdf/motorola_canopy.pdf, 2004. (Last accessed: August 29, 2011)

[14] S. Mustafiz, X. Sun, and J. Kienzle, “Model-driven assessment of system dependability”, Software and Systems Modelling, vol. 7, 2008, pp. 487-502, doi: 10.1007/s10270-008-0084-1.

[15] OWASP, “OWASP Risk Rating Methodology”, https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, 2010. (Last accessed: August 31, 2011)

[16] T. Saridakis, “A System of Patterns for Fault Tolerance”, Procs. of EuroPLoP, 2002, pp. 535–582.

[17] M. Schumacher, E. B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns: Integrating security and systems engineering, Wiley Series on Software Design Patterns, Wiley, 2006.

[18] M. Tichy, “Pattern Based Synthesis of Fault Tolerant Embedded Systems”, Procs. of the Doctoral Symposium of the Fourteenth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), Portland, Oregon, USA, pp. 13-18. ACM Press, November 2006.

[19] Trumba Corporation, “What features does Trumba Connect offer?”, http://www.trumba.com/connect/ knowledgecenter/pdf/Features-list_SS-006.pdf, 2007.(Last accessed: August 29, 2011)

[20] R.Villarroel, E. Fernández-Medina, M. Piattini, “Secure information systems development-A survey and comparison”,

Computers & Security, vol. 24, No 4, pp. 308-321, doi: 10.1016/j.cose.2004.09.011.

[21] N. Yoshioka, H. Washizaki, and K. Maruyama, “A survey on security patterns”, Progress in Informatics, vol. 5, pp. 35-47, doi:10.2201/NiiPi.2008.5.5.