# Security Analysis at Architectural Level in Embedded Software Development

**Liliana Dobrică, Radu Pietraru**

*University Politehnica of Bucharest, Faculty of Automatic Control and Computers, Spl. Independentei, 313, Bucharest, Romania,*
*e-mail:{liliana@ aii.pub.ro, radup@ aii.pub.ro}*

**Abstract:** Beyond the network applications there are increasing concerns regarding reducing risks for the security properties of the final product in the embedded systems domain as early as possible during the development cycle. Current design based on security models proved functional but inefficient when designing embedded software systems. This paper aims to attain the main problems in designing embedded software systems in terms of security and draw a minimal security model specifically designed for them. A systematic analysis of the architecture based on design principles and threats modeling at the conceptual level and model-based at the concrete level is the solution that may guarantee the achievement of security policies specified in requirements.

*Keywords:* security, software architecture, embedded systems, analysis, model

## 1. INTRODUCTION

For many years, embedded systems have been quietly working behind the scenes of almost all modern domains, from automotive to industrial process control to health care and space exploration. Traditionally embedded systems are doing control. They are responsible for the availability and functionality of many critical systems, from factory automation to gas pipeline monitors to networking equipment (Werner *et al,* 2008). Increasingly, these critical embedded systems often incorporate networking requirements and are built from software components and services (Eby *et al,* 2007). Components of embedded systems communicate with each other and with the external environment (e.g, sensors and actuators) at multiple levels (network-on-chip, board-level, cluster-level, local-area networks, wide-area networks) by wire-bound and wireless methods, observing given timing and dependability constraints even in the presence of adverse environmental conditions.

Usually networking refers to control loops to be supported at network level, communication service reliability to be at different levels depending on the application specific parameters, or to follow established protocol standards (Ethernet, USB, CAN, Bluetooth, etc). Integrity and security models and mechanisms are required to prevent undetected modification of hardware and software by unauthorized persons or systems, meaning defence against message injections, message replay or message delay on the network.

Just as the early networked desktop computers and server's were unprepared to address the new security concerns of network connectivity, today's embedded systems introduce a significant new security concern, which must be addressed immediately and systematically. Unfortunately, the critical importance of embedded systems is seldom matched with a strong, comprehensive security infrastructure (Ken Dixon *et al*, 2005). An efficient conceptual design and analysis in the early stages of development reduces these costs of operational resources over-dimensioning of the final product and others related to time and development resources. Also it eliminates the risks that the final product may not be consistent with the functional requirements specification and quality.

Software architecture is the main tangible artifact in a system development cycle. A higher abstraction level of embedded systems design makes no distinctions between software and hardware. Also software architectures provide design-level models and guidelines for composing software systems. The goal of architectural analysis is to get measures of compliance with regard to requirements specification. In the case of embedded systems it is very important to identify which are the relevant properties and how analysis techniques and methods could be applied in this domain. Analysis methods take advantage of architectural concepts to analyze quality attributes in a systematic way. It is very important to identify potential risks and to verify that the quality requirements have been addressed in the architecture design. Analysis could be applied on various stages from the conceptual model to a detailed concrete design. Also it could be associated with the design in an iterative improvement of the architecture. There are specific techniques that could be applied at various development stage and also there are specific costs in time and resources that matter. The potential risk of systems development is minimized if the interactions of quality attributes are considered in the analysis method.

Nowadays security becomes an important property of embedded systems. Novel approaches are required to deal with malicious attacks from intruders, to maintain the confidentiality of sensitive data and to protect intellectual property (IP) and digital rights management (DRM) for embedded systems that are connected to open networks such as the Internet. This paper examines several significant

embedded systems security models that could be considered during a systematic analysis at the architectural level. In the beginning it is discussed about risks analysis in the development process, and then it is outline the principles of secure design, for software security and security mechanisms. Section 4 describes and analysis the main security models and finally we propose a solution of integrating them into a security-based software architecture development for embedded systems. Authors present in section 5 the main steps in security software design for a generic embedded device.

## 2. BACKGROUND

### *2.1. Software Architecture*

Software architecture description is organized in multiple views. Each view is conform to a viewpoint that represents a related set of concerns from a stakeholder perspective. It consists of a model or a diagram that are described with well defined notation such as Unified Modeling Language (UML). Multiple view descriptions of an architecture may be performed at various abstraction levels. Thus it could be a conceptual level with a coarse-grain architectural elements and a concrete level with a detailed design model (Dobrica and Niemela, 2008).

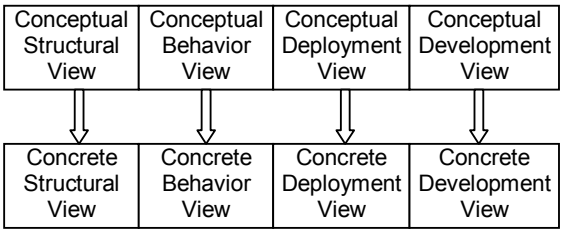


Fig. 1. Software architecture description.

The need for different architectural views depends on three issues: the size, the domain and the number of different stakeholders. Although a multiple view approach helps in developing software products, it is easy to introduce errors and inconsistencies in a multiple view model. It is therefore necessary to provide support for consistency checking among the multiple views.

Architectural styles and patterns are the starting point for architecture development. Architectural styles and patterns are utilized to achieve qualities. A style is determined by a set of component types, the topological layout of the components, a set of semantic constraints and a set of connectors (Bass et al, 1998). A style defines a class of architectures and is an abstraction for a set of architectures that meet it. A small taxonomy identifies five style categories: independent components, data flow centered, data-centered, virtual machine, and call-and return style. An architectural pattern is a documented description of a style or a set of styles that expresses a fundamental structural organization schema applied to high-level system subdivision, distribution, interaction, and adaptation (Buschmann, 1996). Design patterns, on the other hand, are on a lower level. They refine single components and their relationships in a particular context, and idioms describe how to implement particular aspects of components using the given language (Gamma et al., 1994).

Under a software perspective architectural styles are recurring patterns of system organization whose application results in systems with known, desirable properties. As such, styles are key software design idioms. In practice, an architectural style consists of rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among subsystems. The subsystems must comply with the architectural style to avoid a property mismatch at the interfaces between subsystems.

Collections of architecture styles and patterns may be included in a knowledge base that allows their evaluation in terms of both quality factors and concerns, and anticipations of their use. A ”pre-scored” of architectural patterns is feasible in order to gain a sense of their relative suitability to meet particular security of a system. In addition to evaluating individual patterns, it is necessary to evaluate compositions of patterns that might be used in architecture. Identifying patterns that do not compose well (the result is difficult to analyze, or the quality factors of the result are in conflict with each other) should steer a designer away from “difficult” architectures towards those made of well-behaving compositions of patterns. The knowledge base built in this way helps to move from the notion of architectural styles toward the ability to reason (whether quantitatively or qualitatively) based on security-specific models. The goals of having a knowledge base are to make architectural design more routine-like and more predictable, to have a standard set of security-based analysis questions, and to tighten the link between design and analysis by means that can be used to provide context-dependent measures.

### *2.2. Risks in the embedded systems development process*

Many embedded systems require developers to give priority to performance, which in some cases can lead to less priority and consideration of qualities. As most of these systems are conceived and designed for a particular purpose, non-conventional and ad-hoc techniques are often used by developers to introduce new and sometimes unknown classes of threats, not known in general or present in non-embedded devices.

A systematic analysis for embedded systems domain should be related to software components such as:

- Operating System Basic Components
- Memory Management Subsystem
- Third party application programming interfaces
- Bespoke communications protocols

During the activities of a development process (Fig. 2) a comprehensive security review usually is involved in test cases generation, static source code analysis and black box testing. For each one the following risks are considered:

1. Test case generation. Based on an initial threat modeling exercise, test cases are developed that are geared specifically towards embedded systems and its security.
2. Static source code analysis. A combination of static analysis tools for automatic and manual analysis techniques are used to identify potential vulnerabilities in the source code, prioritize components with a higher attack area.
3. Black Box Testing. Manual techniques such as analysis, reverse engineering and fuzzy testing are used to evaluate the system while running and detailed static state without documentation of design or implementation.

A software architecture that integrates security must offer a way by focusing on the analysis of system architecture - providing the ability to detect problems with availability, security, and timeliness early on, before they conspire to raise costs, reduce effectiveness and predictability, and shorten lifespan. Most of these products are expected to do more and be more secure, and many are made to be portable or used in network environments. Software engineers are called on to deliver increasingly complex software systems that provide more functionality while consuming less power and costing less to develop and operate. Unfortunately, system engineers do not have insight into critical system characteristics - such as performance, safety, reliability, time criticality, security or fault tolerance. Using traditional means, system integration becomes high risk, and system evolution (lifecycle support) becomes expensive and results in rapidly outdated components (SEI, 2007).

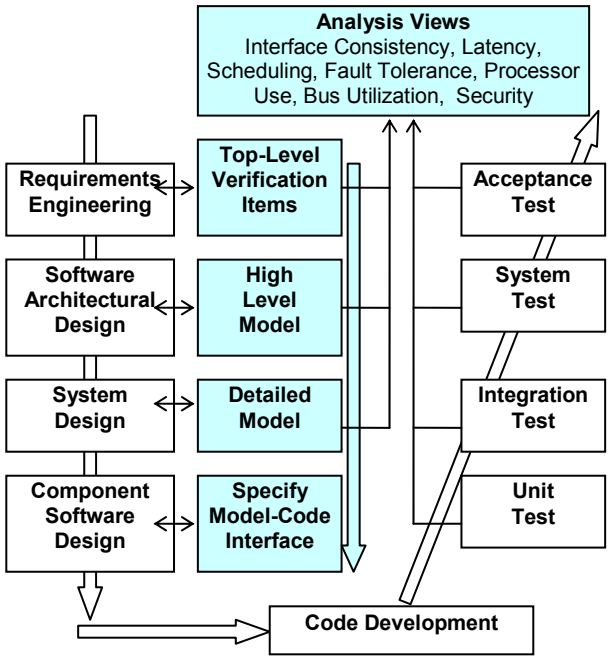


Fig. 2. Integrating security software architecture for embedded system design.

Improved embedded systems engineering practice is architecture-based and model driven. Well-defined software system architecture provides a framework to which system components are designed and integrated. System models that precisely capture this architecture provide the basis for predictable system engineering. Model-based engineering for embedded systems:

- reduces risk through early and repeated analysis of the system architecture;
- permits the engineer to see system-wide impacts of architectural choices;
- increases confidence by validating model assumptions in the operational system and permitting the system models to evolve in multiple fidelity;
- reduces cost through fewer system integration problems and simplified life cycle support;

Integrating security software architecture must permit to:

- represent embedded systems as component based system architecture;
- model component interactions as flows;
- service calls and shared access;
- model task execution and communication with precise timing semantics;
- model the binding of applications to execution platforms;
- represent operational modes and fault tolerant configurations;
- support component evolution and large-scale development;
- accommodate reliability and safety analyses.

## 3. SYSTEMS SECURITY ARHITECTURE

### *3.1. Security requirements of embedded systems*

Embedded systems deal with resource-constrained devices where parameters such as memory capacity, power consumption, processing power, time to market, and cost must strike a balance with security requirements. A synthesis of requirements specification (ARTEMIS, 2005) that refers to security for embedded systems of various application domains includes the following:

R1. The architecture shall assure that authentication is done at link establishment and during execution. Any changes of network addresses and any link modifications have to be detected.
Rationale: There shall be trust and security with respect to network connectivity and the level of trusted embedded environments.

R2. The architecture shall support unique, but uniform identification of the subsystems (at different abstraction levels).
Rationale: Naming is a key topic in networking.

R3. The architecture shall provide mechanisms to prevent the disclosure of information to unauthorized persons or systems.
Rationale: This property is relevant for domain specific information (e.g. personal data in office applications, performance data and product recipes in automation systems)

as well as for information specific to the security mechanisms themselves (e.g. passwords).

R4. The architecture shall provide mechanisms to prevent the disclosure of intellectual property at different abstraction levels (e.g. the contents of an FPGA or of on-chip flash memory).
Rationale: Reverse engineering should be progressively difficult.

R5. The architecture shall provide mechanisms to prevent undetected modification of hardware or software (on platform level and application level) by unauthorized persons or systems.
Rationale: In safety-critical applications, integrity can be a safety-relevant property with respect to information like sensor values or control commands. In many cases the integrity of a given set of data is explicitly required (e.g. mileage counter of car, transaction via online banking).The assurance of integrity includes defence against message injections, message replay, and message delay on the network.

R6. Tuning protection of on-chip Flash memory via a password mechanism with protection option for each flash sector shall be supported. (On chip Flash memories shall support read/program/erase protection option, if the debug ports are active; the activation of the debug port will be disabled by software, the password should be unique for each chip.)
Rationale: In some cases suppliers want to assure that their hardware can be used only with their own software.

R7. The architecture shall provide mechanisms to ensure that unauthorized persons or systems cannot deny service access to authorized users.
Rationale: Violation of availability, also known as denial-of-service may cause economic damages by a significant reduction of the user-value of a given system. Furthermore safety issues can also be affected as operators may lose the ability to monitor and to control a safety-critical process.

R8. The architecture shall provide mechanisms to determine the true identity of a system or a user.
Rationale: Authentication is a prerequisite for authorization.

R9. The architecture shall provide mechanisms to prevent users or systems from accessing a given service without the permission to do so.
Rationale: The basic objective of authorization is to distinguish between legitimate and illegitimate users with respect to a given service.

R10. The architecture shall provide mechanisms to "provide irrefutable proof to a third party of who initiated a certain action in the system, even if the actor is not cooperating".
Rationale: This requirement is relevant to establish accountability and liability. Non-repudiation is becoming extremely important to commerce.

R11. The architecture shall provide mechanisms to bind any content (e.g. music, videos, etc) unambiguously to given license, and to enforce the usage-restrictions (e.g. defined time intervals where usage is allowed) that are stated in the license.
Rationale: This is demanded by digital media publishers in order to allow them to control any duplication and dissemination of their content.
Comment: Component industry: "We want to know how much of our company's IP is in your system".

R12. It shall be possible to protect content that is exchanged via streaming media from being copied.
Rationale: Many content providers want to prevent the copying to disc of a high definition stream with full resolution. Therefore a stream can only be received (decoded) by authorized devices that prevent the digital output of the decoded full resolution stream.

### *3.2. Guidelines and principles of secure design*

Security is a system requirement just like performance, availability, reliability, etc. Therefore, it may be necessary to trade off certain security requirements to gain others during architectural design process. For instance security and dependable architectures are sometimes conflicting. The trade-off between multiple requirements is effective after an analysis at the architectural level to identify sensitivity points where multiple attributes interact. Such an analysis at the conceptual level should be based on the verification of well-known principles from domain knowledge. In embedded systems domain we can identify the principle of threat modeling, principles of secure design, principles for software security and others related to protection mechanisms.

Threat modeling principle is based on the assumption that every system has intrinsic value that is worth protecting. (Giordano, 2009). However, because these systems have value, they are also open to internal or external threats, which can, and often will, cause damage to the end product. Threat modeling can be difficult, but it is necessary. Despite the resource challenges, it is possible to develop a working system that allows a final product to be effective in the presence of adverse environmental conditions by carefully considering a threat model and designing a system to work within the limitations of the available computing power addressing that model. A threat model is defined as identifying a set of possible attacks to consider coupled with a risk assessment strategy. Having a threat model it allows assessing the probability, the potential damage, and the priority of attacks. It involves thinking about how a system can be attacked. When successful, it addresses potential system security failures such as how it fails and what happens when it fails.

Principles of secure design (Savolainen P. *et al*, 2007) provide guidelines at a coarse-grain level for developer to follow such as:

- Design security from the start;
- Allow for future security enhancements;
- Minimize and isolate security controls;
- Employ least privilege;
- Structure the security relevant features;
- Make security friendly;
- Don't depend on secrecy for security.

Principles for software security are associated to kind of techniques that are related to this quality attribute. Examples may be the following:

- Secure the weakest link;
- Practice defense in depth;
- Fail securely, meaning that if the software has to fail, make sure it does it securely;
- Follow the principle of least privilege;
- Compartmentalize, meaning to minimize the amount of damage that can be done by breaking the system into units;
- Keep it simple. Complex design is never easy to understand;
- Promote privacy. Try not to do anything that compromises the privacy of the user;
- Remember that hiding secrets is hard;
- Be reluctant to trust. Instead of making assumptions that need to hold true, you should be reluctant to extend trust;
- Use your community resources. Public scrutiny promotes trust;

Design principles for protection mechanisms are:

- Least privilege - should only have the rights necessary to complete your task.
- Economy of mechanism - should be sufficiently small and as simple as to be verified and implemented – e.g., security kernel. Complex mechanisms should be correctly understood, modeled, configured, implemented and used.
- Complete mediation - every access to every object must be checked.
- Open design- let the design be open. *Security through obscurity* is a bad idea.
- Should be open for scrutiny by the community - better to have a friend/colleague find an error than a foe.
- Separation of privilege - access to objects should depend on more than one condition being satisfied.
- Least common mechanism - minimize the amount of mechanism common to more than one user and depended on by all users.
- Psychological acceptability - user interface must be easy to use, so that users routinely and automatically apply the mechanisms correctly. Otherwise, they will be bypassed.
- Fail-safe defaults- should be lack of access.

## 4. SECURITY MODELS

A more detailed analysis of security at the architectural level should consider building of specific security models. Security models are built based the security policies, which are documented and express clearly and concisely what the protection mechanisms are to be achieved. More concrete a policy is a statement of the security we expect from the final system. Thus a security model is a specification of a security policy and it mainly describes the entities governed by the policy and it states the rules that constitute the policy (Yu H. *et al*, 2005).

A security model maps the abstract goals of a policy to the embedded system terms by specifying explicit data structures and techniques that are necessary to achieve the security policy. A security model is usually represented in mathematics and analytical ideas, which are then mapped to system specifications, and then developed by programmers through programming code. For example, if a security policy states that subjects need to be authorized to access objects, the security model would provide the mathematical relationships and formulas explaining how x can access y only through the outlined specific methods. A security policy outlines goals without regard to how they will be accomplished (Feng Q. *et al*, 2005). A model is a framework that gives the policy form and solves security access problems for particular situations.

There are various types of security models. We can mention the following:

- Models that capture policies for confidentiality (Bell-LaPadula) or for integrity (Biba, Clark-Wilson).
- Models that apply to environments with static policies (Bell-LaPadula) and models that considers dynamic changes of access rights (Chinese Wall).

Security models can be informal (Clark-Wilson), semi-formal, or formal (Bell-LaPadula, Harrison-Ruzzo-Ullman).

Several models are described bellow. Our discussion refers to lattice models, state machine models, noninterference models, Bell-LaPadulla confidentiality model, Biba integrity model,

### *4.1. Lattice Models*

A lattice is a mathematical construct that is built upon the notion of a group. A lattice is a mathematical construction with:

- a set of elements
- a partial ordering relation
- the property that any two elements must have unique least upper bound and greatest lower bound

A security lattice model combines multilevel and multilateral security. Lattice elements are security labels that consist of a security level and set of categories.

### *4.2. State Machine Models*

In state machine model, the state of a machine is captured in order to verify the security of a system. A given state consists of all current permissions and all current instances of subjects accessing the objects. If the subject can access objects only by means that are concurrent with the security policy, the system is secure. The model is used to describe the behavior of a system to different inputs. It provides mathematical constructs that represents sets (subjects, objects) and sequences. When an object accepts an input, this modifies a state variable thus transiting to a different state.

*4.3. Noninterference Models*

The model ensures that any actions that take place at a higher security level do not affect, or interfere with, actions that take place at a lower level. It is not concerned with the flow of data, but rather with what a subject knows about the state of the system. So if an entity at a higher security level performs an action, it can not change the state for the entity at the lower level. The model also addresses the inference attack that occurs when some one has access to some type of information and can infer(guess) something that he does not have the clearance level or authority to know (WikiBooks, 2009).

*4.4. Bell—LaPadula Confidentiality Model*

It was the first mathematical model with a multilevel security policy that is used to define the concept of a secure state machine and models of access and outlined rules of access.
It is a state m/c model that enforces the confidentiality aspects of access model. The model focuses on ensuring that the subjects with different clearances (top secret, secret, confidential) are properly authenticated by having the necessary security clearance, need to know, and formal access approval-before accessing an object that are under different classification levels (top secret, secret, confidential).
The rules of Bell-Lapadula model are:

- Simple security rule (no read up rule): It states that a subject at a given security level can not read data that resides at a higher security level.
- Star property rule (no write down rule): It states that a subject in a given security level can not write information to a lower security levels.
- Strong star property rule: It states a subject that has read and write capabilities can only perform those functions at the same security level, nothing higher and nothing lower.

*4.5. Biba Integrity Model*

It is developed after Bell – Lapadula model. It addresses integrity of data unlike Bell – Lapadula which addresses confidentiality. It uses a lattice of integrity levels unlike Bell – Lapadula which uses a lattice of security levels. It is also an information flow model like the Bell – Lapadula because they are most concerned about data flowing from one level to another. The rules of Biba model are:

- Simple integrity rule(no read down): it states that a subject can not read data from a lower integrity level.
- Star integrity rule(no write up): it states that a subject can not write data to an object at a higher integrity level.
- Invocation property: it states that a subject can not invoke(call upon) a subject at a higher integrity level.

*4.6. Clark—Wilson Integrity Model*

It was developed after Biba and addresses the integrity of information. This model separates data into one subject that needs to be highly protected, referred to as a constrained data item (CDI) and another subset that does not require high level of protection, referred to as unconstrained data items(UDI).

Components of the model are:

- Subjects (users): are active agents.
- Transformation procedures (Tp's): the s/w procedures such as read, write, modify that perform the required operation on behalf of the subject (user).
- Constrained data items (CDI): data that can be modified only by Tp's.
- Unconstrained data items (UDI): data that can be manipulated by subjects via primitive read/write operations.
- Integrity verification procedure (IVP): programs that run periodically to check the consistency of CDIs with external reality. These integrity rules are usually defined by vendors.

Integrity goals of Clark – Wilson model are

- Prevent unauthorized users from making modification (addressed by Biba model).
- Separation of duties prevents authorized users from making improper modifications.
- Well formed transactions: maintain internal and external consistency i.e. it is a series of operations that are carried out to transfer the data from one consistent state to the other.

*4.7. Access Control Matrix*

This model addresses access control issues. Commonly used in operating systems and applications.

*4.8. Information Flow Models*

In this model, data is thought of as being held in individual discrete compartments. Information is divided based on two factors: classification and need to know

The subjects clearance has to dominate the objects classification and the subjects security profile must contain the one of the categories listed in the object label, which enforces need to know.

*4.9. Graham—Denning Model*

This model defines a set of basic rights in terms of commands that a specific subject can execute on an object. It proposes the eight primitive protection rights, or rules of how these types of functionalities should take place securely.

- How to securely create an object.
- How to securely create a subject.
- How to securely delete an object.
- How to securely delete a subject.
- How to provide read access rights.

- How to provide grant access rights.
- How to provide delete access rights.
- How to provide transfer access rights.

*4.10. Harrison—Ruzzo—Ullman Model*

The HRU security model (Harrison, Ruzzo, Ullman model) is an operating system level security model which deals with the integrity of access rights in the system. The system is based around the idea of a finite set of procedures being available to edit the access rights of a subject s on an object. The model also discussed the possibilities and limitations of proving safety of a system using an algorithm.

*4.11. Brewer—Nash (Chinese Wall)*

This model provides access controls that can change dynamically depending upon a user's previous actions. The main goal of this model is to protect against conflicts of interests by user's access attempts. It is based on the information flow model, where no information can flow between subjects and objects in a way that would result in a conflict of interest. The model states that a subject can write to an object if, and only if, the subject can not read another object that is in a different data set.

## 5. EXAMPLE OF INTEGRATED SECURITY ANALYSIS IN ARCHITECTURE DESIGN

An architecture development method can be applied iteratively integrating design and analysis techniques.

Based on this the architecture description is revealed more detailed and improved until the requirements are realized on the architecture design and it can proceed with the implementation. Security concerns that have been introduced previously are integrated in this method as in the following figure (Fig. 3). Architecture description at conceptual level integrates security guidelines and the architecture description at the concrete level allows analysis based on security specific models.

After this phase, the activities involved in concrete architecture design and the second phase of architecture analysis are performed.

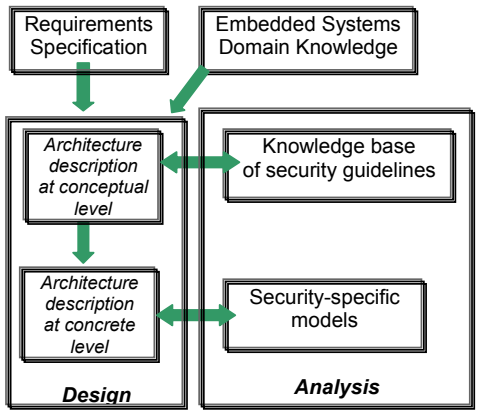


Fig 3. Iterative architecture development.

For a generic embedded device with such requirements specification an iterative architecture development must follow the next software design and analysis steps:

(1) draw the main functions of the device in a conceptual level architecture description;

(2) analyze the risks of the conceptual level description based on security guidelines;

(3) describe the architecture at a more concrete level;

(4) analyze risks on the architecture design by modeling the threats for the real device;

(5) identify and implement in the concrete design the measures to reduce the risks;

(6) at the end of the development cycle re-analyze the level of the final risks and remodel the threats;

(7) in a next generation architecture design re-use from the domain knowledge an architecture and the models of security and the feedback from the real world to build a better architecture.

## 6. CONCLUSIONS

Security system requirements and activities for realizing security should be considered in the early stages of development. In the context of complex embedded system an architecture-driven approach is the solution that provides the quality of the final system. A systematic analysis of the architecture based on design principles at the conceptual level and model-based at the concrete level is the solution that may guarantee the achievement of security policies specified in requirements. In this paper we have discussed about the main problems in designing embedded software systems with security requirements and we have presented the current design and analysis techniques that could be applied at the architectural level.

## ACKNOWLEDGEMENTS

This work is supported by the Romanian research grant CNCSIS IDEI no. 627/2009.

## REFERENCES

Eby M., J. Werner, G. Karsai, A. Ledeczi, (2007) Embedded Systems Security Co-Design, Institute for Software Integrated Systems.

Feng Q., R. Lutz, (2005) Bi-directional safety analysis of product lines, *Journal of Systems and Software*, 78, pp. 111-127.

Ken Dixon, ConnectForte, Jerry Krasner (2005) FIPS 140-2 and the Embedded Marketplace: Does FIPS Really Matter?

Dobrica L., Niemelä E. (2008), A UML-based variability specification for product line architecture views, Procs. of the Third International Conference on Software and

Data Technologies (ICSOFT 2008), vol. SE, INSTICC Press, pp. 234-239.

Bass L., P. Clements, and R. Kazman, (1998) Software Architecture in Practice. Addison-Wesley.

Buschmann F., R. Meunier, and H. Rohnert, (1996) Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons.

Gamma E., R. Helm, R. Johnson, and J. Vlissides, (1994), Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

Sachitano A., R.O. Chapman, and J.A. Hamilton, (2004) Security in Software Architecture: A Case Study.

Savolainen P., E. Niemelä, R. Savola, (2007) A Taxonomy of Information Security for Service Centric Systems, *Procs. of Euromicro-SEAA*, 2007, pp.5-12.

Werner J., M. Eby, J. Mathe, G. Karsai, Y. Xue, J. Sztipanovits, (2008) Integrating Security Modeling in Embedded System Design, Institute for Software Integrated Systems, Vanderbilt University, Nashville.

Software Engineering Institute - http://www.sei.cmu.edu/ (2007) - Predictable, Model-Based Engineering for Embedded Systems.

Yu H., X. He, Y. Deng, L. Mo, (2005) Integrating Security Administration into Software Architectures Design.

ARTEMIS Strategic Research Agenda (2005), <http://www.artemis-office.org/DotNetNuke/Portals/0/Documents/sra.pdf>

Giordano Philip, (2009), How Secure is Secure for Embedded Systems' Design, Analog Devices, Inc.

WikiBooks - http://en.wikibooks.org (2009) - Security Architecture and Design