Architectural tactics for security and their realizations

Eduardo B. Fernandez[1], Hernán Astudillo[2], and Gilberto Pedraza-García[3,4]

[1] Dept. of Electrical and Computer Eng. And Comp. Science, Florida Atlantic University, Boca
Raton, FL, USA, fernande@fau.edu
[2] Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile, hernan@inf.utfsm.cl
[3] Universidad de Los Andes, Bogotá, Colombia, g.pedraza56@uniandes.edu.co
[4] Universidad Piloto de Colombia, Programa de Ingeniería de Sistemas, Bogotá, Colombia.

Abstract. Building secure systems is a difficult task and several approaches using different artifacts have been proposed for handling it, including tactics and security patterns. Architectural tactics are design decisions intended to improve some system quality factor. Since their initial formulation, they have been formalized, compared with patterns, associated to styles, and several realizations have been considered. Until now, tactics have been applied only by software architects, we thought that they should be analyzed from a security perspective. We have examined the initial tactics set and classification from the viewpoint of security research, and concluded that some tactics would be better described as principles or policies, some are unnecessary, and others do not cover the functions needed to secure systems. This article proposes a reasoned examination, pruning and reclassification of architectural tactics for security, and argues why it is more appropriate than the original and the previously refined sets. We justify our conclusions through an example, literature search, and logical reasoning. We consider the realization of the modified set of tactics using security patterns. We present also an extension of tactics to make them more usable in practice. In addition to our specific methodological contribution, we aim to enrich the conversation of the software architecture and security communities.

Keywords: Architecture tactics; secure architectures; security patterns; secure software development.

1. Introduction

Secure systems are notoriously hard to build; like most global system quality criteria, a piecemeal approach based on securing individual system units does not produce a secure system; attackers can find vulnerabilities in other parts of the system and combining components which are secure individually may introduce new vulnerabilities. Many approaches to build secure systems have been proposed, but they usually focus on some specific aspect, e.g. authorization, and address only a few types of threats. Design decisions about incorporating security mechanisms have a global effect on security as well as on other quality attributes, e.g. availability and scalability, and thus local optimizations are not possible. From a security research standpoint, lacking

quantitative measures, a secure system is one that can be shown to withstand a variety of attacks or that satisfies a list of security requirements.

The security research community has been studying how to build secure systems for a long time but has produced few studies or methods to make a whole system secure [14, 40, 58]. On the other hand, the software architecture literature addressed security as one of several global quality properties, proposing to use "architectural tactics" [5, 6] and emphasizing global aspects [46,64]; however, the tactics currently presented in the literature are not justified on coverage or parsimony grounds. Also, security tactics give general guidance but not detailed construction advice. Since their initial formulation, tactics have been formalized [3], compared to patterns [33], related to the Common Criteria [43], and associated to architectural styles [1, 27]. The initial set of tactics for security [5, 6] has been refined once [48] but later work ignored the proposed new set. While apparently accepted by software architects, there is no written record of the use of tactics in practice and only a few publications (see Section 2) discuss their use for designing secure systems; there are no arguments or analysis either to justify this specific set. Tactics are not mentioned in any of the best-known secure development methodologies [58] but we think that they can contribute to improve such methodologies, which motivates our interest in them. We need effective ways to build secure systems and we believe that by combining tactics and patterns we can improve secure software development methodologies; but we first need to revise the existing tactics because we think they have some aspects which prevent them from being truly useful for security. We attempt to combine security approaches from security and software architecture researchers, which we believe is in itself an important goal.

Software architecture researchers separate architecture from the application or system requirements, while security methodology researchers define a complete secure lifecycle. In addition, requirements researchers (a third group) focus on requirements and leave the architecture and final development steps to others. This separation results in disjoint work, but security requires an integrated approach along all stages and across all architectural levels; we consider this integration a fundamental objective of our approach.

This article presents a reasoned examination, pruning and reclassification of architectural tactics for security, considering both the original set and the revised set and applying domain-specific security knowledge. We also address tactics utilization by designers, describing their possible realization with security patterns. *Patterns* are encapsulated solutions to recurrent problems in specific contexts, *security patterns* define solutions to handle threats, to fix a vulnerability, or to realize a security property [16]. Patterns are considered a good way to build secure systems and several methodologies based on them exist [13, 16, 58, 59]. Patterns include several sections that define in addition to a solution, their use, context, applicability, advantages, and disadvantages. Tactics have also been defined for other software architecture quality factors such as reliability, availability, and safety, which are also important, but we

concentrate on security in this paper. Several secure software development methodologies using patterns have been proposed; one of them (ASE) was proposed by us [59] and we use ideas from it in our examples. Other realizations for tactics have been proposed but as we discuss later we consider patterns the most promising approach. For those cases where using patterns is not possible or convenient we propose an intermediate realization using a template to describe tactics, which presents enough detail to help experienced software architects.

Tactics and patterns are the result of experience building and using secure systems and although they can be formalized, they do not have a theoretical basis, only a logical and empirical justification, they are abstractions of best practices from real systems. As such, we cannot formally prove that our new tactics are complete or optimal in some sense. In fact, except for small systems, there is no way to build provable-secure systems [35]. There are no general measures of security either [37, 52]. We are interested in complex general-purpose systems, not on specialized or restricted systems.

This article is an extension of our ECSA paper [19] and its contributions include:

- A revised set of tactics, based on an extensive search of security knowledge, which is our main contribution. A more effective and usable set of tactics has a clear practical value.
- A detailed consideration of the use of security patterns as a way for realizing tactics, including an illustrative example. We use ASE as a framework to be complemented by the use of tactics and patterns. This example is also an aspect of the validation of our ideas.
- We show that tactics have a role not only in the design stage (architecture) but also in the conceptual model that results from the requirements, thus helping connect two frequently disjoint areas.
- A realization for tactics in the form of a template that can be used when security patterns are not convenient.
- A discussion of the correspondence of some security and software architecture concepts to understand them better. As indicated, the security and software architecture communities are rather disjoint and we attempt to help bridge their gap.

The remainder of the article is organized as follows: Section 2 describes the use of architecture tactics to build secure systems; Section 3 discusses security patterns; Section 4 defines security principles and policies, and other terms used for secure systems design; Section 5 examines the initial (and still used) tactics tree and indicates its problems; Section 6 presents our new tactics based on security knowledge; Section 7 proposes a realization for tactics using security patterns; Section 8 presents an example of the use of the new tactics. Section 9 proposes a template for describing tactics. Section 10 discusses related work, Section 11 considers validation approaches, and Section 12 provides some conclusions and ideas for future work.

2. Architectural tactics to build secure systems

Architectural tactics, originally introduced to the literature in 2003 [5], are "measures" or "decisions" taken to improve some quality factor, a definition later refined to "architectural building blocks from which architectural patterns are created" [6]. Each tactic corresponds to a design decision with respect to a quality factor; i.e., tactics codify and record best practices for achieving that factor.

Rozanski and Woods [46, 64] defined architectural tactics as architectural design guidance, i.e. strategies or advice on how to drive a general design issue related to improving required quality attributes without imposing a particular software structure. They also suggest (but give little operational detail) that the application of security tactics may be expressed in the software architecture as adding, modifying, or deleting architectural elements with specific responsibilities, introducing security technologies, or describing new operational procedures to support secure operation.

The literature records a few approaches to use architectural tactics to build secure systems. Harrison and Avgeriou [27] proposed (see Figure 1) that the architecture be first defined using architecture patterns to determine the structural aspects of the functional requirements, and then apply tactics to introduce non-functional aspects such as security and reliability. Their work does not attempt to evaluate the actual level of security or reliability thus obtained, or whether different realizations may yield unnecessary security mechanisms. In fact, this work does not indicate any realization of the selected tactics. They do not consider specific threats either and assume that the security requirements have already been defined. However, this paper is until now is the most comprehensive treatment of how to apply security tactics.

In Woods and Rozanski [64], security tactics are part of a wider framework to improve quality attributes, which defines a set of activities, checklists, tactics and guidelines. It also suggests that the impact of security tactics in software architecture may be described on architectural views (context, functional, information, concurrency, development, deployment, and operational). In order to achieve security quality attributes in software architecture, they propose to identify sensitive resources, define the security policy, identify threats to the system, design the security implementation, and assess the security risks. These steps are indeed close to a security methodology, such as the one described in [59]; although they do not describe how architects can apply the security tactics catalog, it is a good basis for a more complete security methodology.

The Attribute Driven Design (ADD) method is an approach to defining a software architecture based on the expected quality attributes of the software [4]. It is a recursive decomposition process where, at each stage in the decomposition, attribute primitives are chosen to satisfy a set of quality scenarios and then functionality is allocated to instantiate the component and connector types provided by the primitives. The quality

attributes are implemented as patterns, although there are no details of these patterns and only five patterns are mentioned. The SEI produced also a detailed example of applying reliability tactics, which have some relation to security tactics [63]. A report from SEI discusses security and survivability aspects of systems [12]. Some work exists also on the realization of security tactics, which we discuss in Section 7.

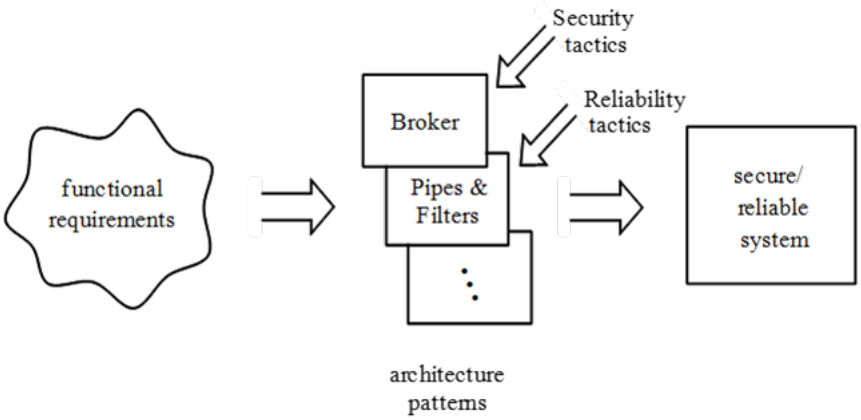

Fig. 1. Producing secure or reliable architectures from tactics.

3. Building secure systems with patterns

As indicated, patterns are encapsulated solutions to recurrent system problems and define a vocabulary that concisely expresses requirements and solutions without getting prematurely into implementation details [8]. Most security patterns are a type of architecture pattern in that they describe global software architecture concerns, although some of them can be considered to be a type of design[1] pattern as well. Finally, some security patterns are a type of analysis pattern in the sense that security constraints should be defined at the highest possible (semantic) level of the system where no implementation details are present to obscure conceptual solutions [15]. Because of their abstraction properties, security patterns provide a way to apply a holistic approach to system security and they are very valuable to handle large and complex systems.

Design patterns are increasingly, but slowly, being accepted in industry; Microsoft, Siemens [8], Ericsson [31], Sun (Oracle), Motorola, and IBM, among others, have reported their use and have web pages and even books about them. Security

[1] We consider standard design patterns as code oriented.

patterns also are starting to be accepted, with some companies producing catalogs in books and in the web, including IBM [30], Sun (Oracle) [55], Ericsson [59] , General Dynamics, and Microsoft [38]. In particular, our patterns have appeared in industrial catalogs and have been used in industrial projects, including a large Spanish bank (BBVA) [39], and two Japanese companies (Fujitsu and Toshiba). Hafiz and Johnson describe the improvements to a complex industrial software architecture using security and reliability patterns [26]. The Tizen and Firefox operating systems include several security patterns [28].

While there is no "official" template for security patterns, we use a variation of the POSA template [8], which is composed of a thumbnail of the problem it solves (a threat to the system), the context where the pattern is applicable, a brief description of how it solves the problem (stops or mitigates the threat), the static structure of the solution (usually UML class diagrams), the dynamic structure of the solution (usually UML sequence diagrams or possibly activity or state diagrams), and guidelines for the implementation of this pattern. The contents of all the sections of the template are fundamental for the correct use of the pattern. Patterns are not just solutions and are not plug-ins or software components but are suggested solutions that must be tailored to fit specific applications.

The effect of a pattern on security, performance, or any other factor depends on how it is used; for example, applying authentication in many places in a system may increase security but reduces performance if users or processes need to re-authenticate themselves each time they access resources. In fact, authentication in the wrong places does not even increase security. There are tradeoffs when improving any quality factor as well as more than one approach on how to do it; for example, "Encrypt data" can be realized in more than one way, i.e., symmetric or asymmetric encryption. Depending on the application, one realization can be more convenient than another. To make things worse there are several varieties and ways to apply either type of encryption. If architects are not experienced or knowledgeable about security, they need more detailed guidance than just saying "Encrypt data", we observe that this is especially true for system developers, who are usually not experts on security. Security patterns can improve the use of tactics by providing more details that can help select and apply the right tactic in a particular situation.

An extensive literature exists on how to build secure systems using patterns, and other types of artifacts such as aspects [58]. Systematic approaches, particularly those implying some sort of process aligned with the software development life-cycle, are called *security methodologies*. There are a number of security methodologies in the literature, of which the most flexible and most satisfactory from an industry adoption viewpoint are methodologies that encapsulate their security solutions in some fashion, such as via the use of security patterns. We have proposed one of these, ASE, described in [59]. ASE considers first the use cases of a system during Requirements analysis, and how these use cases can be subverted (Security Requirements Determination (SecReq) phase, Adversary Modeling (AdvMod) stage). This leads to an identification of countermeasures appropriate at this early stage of development and their modeling into the system's conceptual model. A similar sequence of activities is

performed during Design, this time with respect to the system's architecture, considering in detail areas of functionality, threats, appropriate countermeasures, and then modeling the countermeasures to produce a secure software architecture. Patterns are applied using *security solution frames*, which are structured groups of related patterns. The remaining activities are concerned with verification (at different stages) and security implementation, which usually implies selecting COTS components that realize mechanisms, e.g. firewalls or using security frameworks [10] ASE is an improvement on our early work [13]. Other methodologies using patterns exist and their stages are similarly structured following a software lifecycle.

4. Security Principles and Policies

Security differs from other quality factors, like availability or scalability, in its close tie to the application or system semantics; e.g., "only the owner of an account can withdraw money from it", or "a process cannot write outside its own virtual address space". Also, external regulations may prescribe (directly or indirectly) specific information protection needs, leaving little space for possible tradeoffs among tactics. Before analyzing tactics we need to discuss first some related concepts.

A dictionary definition of principle is: "a fundamental truth or proposition that serves as the foundation for a system of belief or behavior or for a chain of reasoning". *Design principles* are fundamental truths or assertions that define guidelines to produce good designs. Saltzer and Schroeder's classic paper [50] defined a set of *security principles*, including among others: least privilege, separation of duty, and least common mechanism (two detailed examples are shown below). Additional design principles have been added with time, like "defense in depth" and "start from semantic levels" [53]. Indeed, Neumann [40] considers the use of principles fundamental to produce secure systems. Principles are very general and may have many possible realizations; they must be followed (explicitly or implicitly) when building systems to infuse security in them, but their unaided application depends heavily on the designer knowledge and experience.

Policies are high-level guidelines defining how an institution conducts its activities in its business, professional, economic, social, and legal environment [16]. A similar definition is [51]: A policy is typically a document that outlines specific requirements or rules that must be met. *Regulations* are local or government policies that must be reflected in the implemented system, e.g., HIPAA applies to applications that handle medical records [28]. Industry regulations are usually called *standards;* for example WS-Security and SAML are standards for the security of web services. Regulations and standards can refer to security or to other non-functional requirements such as availability or scalability. While a regulation implies the obligation of following them (because of the authority of a country or state), a standard is usually a recommendation from industry groups.

The *institution security policies* include laws, rules, and practices that regulate how an institution uses, manages and protects resources. In the information/network secur-

ity realm, policies are usually point-specific, covering a single area. For example, an "Acceptable Use" policy would cover the rules and regulations for appropriate use of the computing facilities [51]. Another example of an institution policy is [56]: “Computer security responsibilities and accountability should be made explicit”

Operationally speaking, policies are management instructions indicating a predetermined course of action or a way to handle a problem or situation. Every institution has a set of policies, explicit or implicit, some of which are security policies. Institution security policies are essential to build secure systems since they indicate (1) what is important to protect and (2) how much effort and money to invest in this protection.

At the system level, policies may also indicate a preferred way to avoid or mitigate threats; e.g., a mutual authentication policy avoids impostors from either side, indicating that authenticity is important to the institution. Every system uses a combination of policies according to its objectives and environment. As an example, two of the most common system security policies used in practice are:

- *Open/closed systems* —in a closed system, nothing is accessible unless explicitly authorized, whereas in an open system everything is accessible unless explicitly denied. Institutions where information security is very important, such as banks, use a closed policy (e.g., “only an account’s owner can access it”); institutions whose objective is disseminating information, such as libraries, use an open policy (e.g., “all books are accessible except those labeled as rare books”).
- *Least privilege (need to know)* —people or any active entity that needs to access computational resources must be given authorization only for those resources needed to perform their functions (e.g., “a secretary should only have access to employees’ names and addresses”); it also applies to the institution (e.g., “it should not collect more information than strictly necessary about its members”). This policy can be considered a refinement of the closed system policy as well as a principle.

Policies are prescriptive, and can be thought of as directions for designers, a possible definition emphasizing this is: “a strategic process mandate that establishes a desired (security) goal” [21]. Policies can be structured as hierarchies, and more specific policies may apply to the components of a system. It is possible to express individual policies using UML class diagrams with constraints. In software architecture terms, policies are guidelines to apply tactics, which in turn may be realized using for example security patterns, as shown in Figure 2. In this model we consider only system policies, not regulations or standards. The associations are many-to-many: a policy may be applied using several tactics; for example, a Mutual Authentication policy requires two tactics: Identify Actors and Authenticate actors, but Identify Actors is also used in Maintain Audit Trail. Similarly, tactics and security patterns have a many to many relationship: the tactic “Maintain confidentiality” can be obtained using Cryptographic security patterns or using Authorization patterns. Inversely, Cryptographic patterns can be used to Limit Access. Principles can be applied in building policies, tactics, and patterns; in fact, the lack of principles is a principle in itself. For example,

need-to-know (least privilege) is a common principle that correspond directly to policies in many institutions, where employees are given only the rights they need to perform their jobs. This principle can also be used in defining tactics to reduce exposure since now fewer users can access the system and only in limited ways; it can finally be used in the definition of Authorization patterns to have a finer granularity access control.

As another example, Figure 3 (a UML class diagram with annotations) indicates a hierarchical policy which prescribes that "only owners of accounts can access their accounts", which is translated into two more specific policies, for *Authentication* and for *Content-Dependent Authorization*, which can be realized by corresponding security patterns. In this example, the "content-dependent authorization" policy can prescribe the use of the tactic "Authorize actors", which would be realized by a "Content-dependent Authorizer" security pattern [18]. Broad policies or tactics usually require applying several patterns for their realization; e.g. "Authorize actors" can be obtained by combining the patterns *Authenticator*, *Authorizer*, and *Security Logger/Auditor* [16].

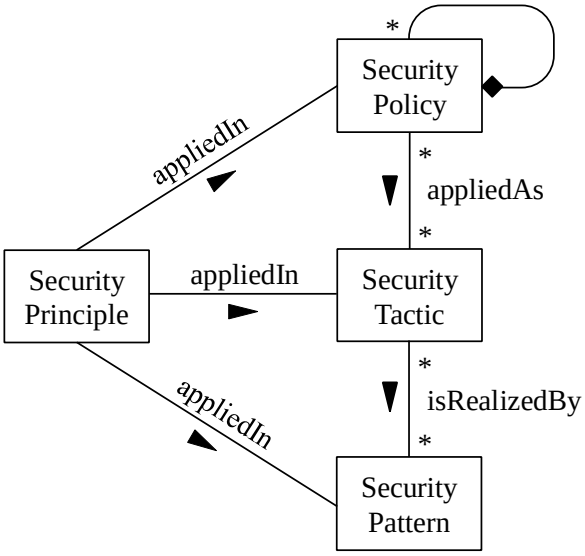

Fig. 1. From policies to security patterns.

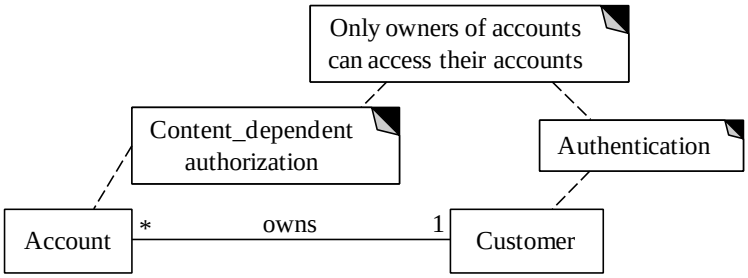

Fig. 1. Hierarchies of policies.

5. Evaluating the current tactic set from a security standpoint

Good practices of security design require that security be enforced at the system (platform) level, not by individual applications [2, 25, 42]. By application we mean any software with some functional business purpose, e.g., a financial or a manufacturing software. By system or platform we mean all the supporting system levels, including the hardware, the operating system, one or more database management systems, and distribution middleware. Since several applications may share a common platform, policies about resource access must cover all applications sharing system resources, e.g., all of them should share mechanisms for intrusion detection, authentication, authorization enforcement, and logging; i.e., all the standard security defenses. Additional security constraints may be placed within applications but a core set of security mechanisms controls the execution of all applications. This separation is important even if there were just one application on the platform, to decouple those aspects which are not intrinsic to a specific application and to foresee possible evolution (or new applications). Incidentally, this is the same motivation behind other security encapsulation approaches like aspect-oriented design [44]. A practical implication is that architects only need to verify that the system implements the tactics appropriate for their desired degree of security, instead of including them in the application-specific design; they need to add invocations to the corresponding services but not actually build those security services, which, as mentioned earlier, are usually implemented as COTS components.

The original list of tactics is structured as a classification tree (see Figure 4); the tactics are the tree leaves and most of them are at the same level, i.e., this is not strictly a tree.. Although security patterns can be classified [61, 62] to cover all concerns, all the architectural levels of a system, and other facets with a multidimensional matrix, for tactics we will keep their tree structure due to their small number. The branches of the current tree correspond to one of the dimensions in [61]. And we have changed “Resist attacks” to “Stop or mitigate attacks”, which is closer to what security designers try to accomplish. We start by removing some tactics considered not useful and

later we add some new ones. We do not indicate here how to realize tactics, but address this in Section 7.

Some security tactics in Figure 4 are actually security principles. As indicated in Section 3, principles are not specific enough to become patterns or even tactics; there may be millions of solutions that satisfy a given principle[2]. Thus, tactics that correspond to principles are not operationally useful; they are good general recommendations, but the designer gets no concrete guide about their realization, nor a reason to apply these principles and not others. For example, one can reduce exposure by reducing the number of explicit or implicit inputs (reducing its attack surface []), by decoupling or merging units of the application, hiding operations using layering, and in other ways. "Separate entities" can be done using virtual machines [16], sandboxed domains [16], or cryptography. We can then eliminate tactics "Limit exposure", "Limit access", and "Separate entities"[3]. This does not mean we are not including principles, but they are implicit when using well-thought patterns, which are more concrete and easier to use for designers. Additionally, the selection of principles in the original set is also rather arbitrary; for example, why not "apply least privilege (need to know)" which is a fundamental security principle?

[2] Kim tried to implement this kind of tactics but his architectural blocks chose one specific implementation which precluded other design possibilities [34].

3 A further confirmation of the lack of use of these tactics is shown in [49], where in their study of 53 open source software projects none of them used them.

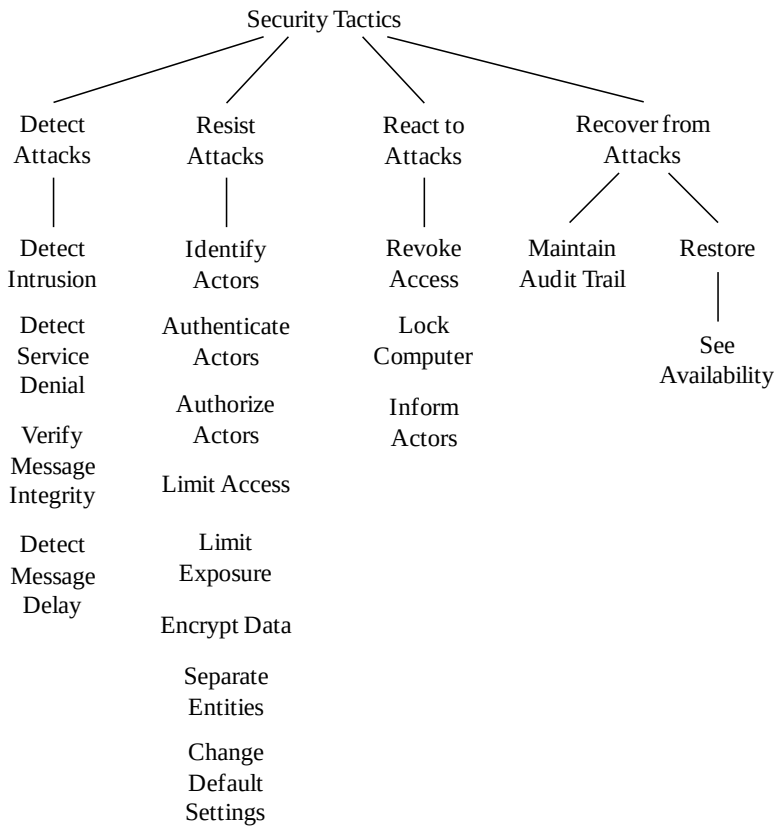

Fig. 1. Classification of security tactics according to Bass et al. [3].

We can also eliminate “Detect message delay”, which is a way to detect some attacks, but if we include it would also need to include “detect abnormal behavior”, “match traffic events to known attacks”, and other approaches used by IDSs []. In other words, this is really a specific way to detect attacks. DoS (Denial of Service) is just another type of attack, so “Detect service denial” is included in “Detect intrusion”; Intrusion Detection products can usually detect DoS. “Identify actors” is not a tactic to resist attacks, it is a more fundamental and general mechanism in distributed systems, necessary to implement authentication, authorization, secure channel, logging, and recovery..

As indicated, some of the proposed tactics are functions that apply to all applications, not to specific application architectures; as such, they don’t need to be incorporated in each individual application. Some like “Verify message integrity” could be left in the set of tactics if the application needs to have its own way of applying verification checks but it is much more likely to be implemented in the platform as a shared service . Which security mechanisms remain in the application and which ones are left to the system requires experience and depends on the specific application; incor-

porating these tactics in all applications is not efficient and can lead to inconsistencies in system behavior.

Similarly, tactics for reacting to attacks should include system functions, which are implemented independently of any application, as well as application-specific policies. “Maintain audit trail” is clearly a system function that can be used to detect attacks and to recover from attacks, although we may perform logging in a way specific to an application. “Change default settings” is about reconfiguration and boot up; these are operational system functions, and do not belong in an application design (although in some applications they can make sense depending on what settings are being changed). “Lock computer” may not always be possible, e.g., in a flight control system we would instead to reconfigure it to work in degraded mode; it should be done by a specific application in any case. Finally, we eliminated “Change default settings” because it is a system-operational function, not advice for designers.

6. New or Modified Tactics

As a design principle, a tactic should not be neither too general nor too specific: if too general, the designer gets confused by the variety of alternatives (the reason to eliminate principles), and if too specific, designers get mechanisms instead of tactics, and their options are restricted *a priori*.

Figure 5 describes the modified set of security tactics, which are organized in four branches, similarly to the original set: detect attacks, stop or mitigate attacks, react to attacks, and recover from attacks. We describe each one of these branches below.

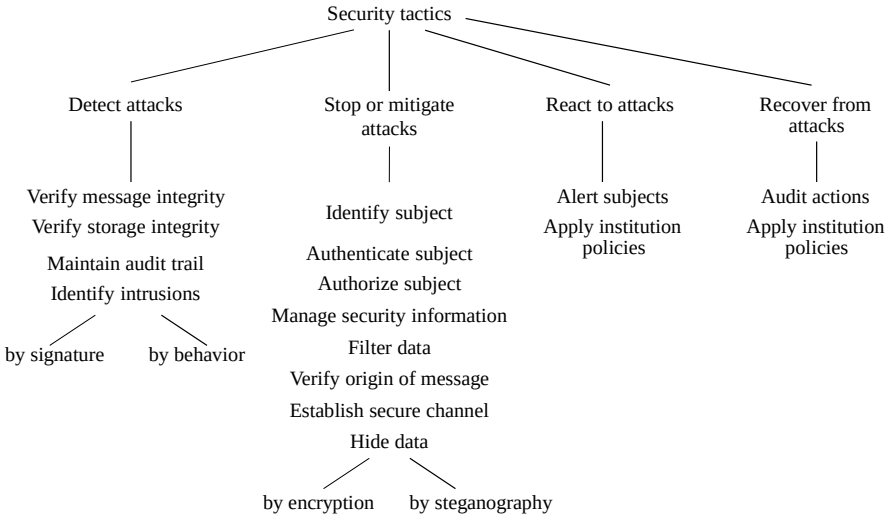

Fig. 5. A new set of tactics.

6.1 Detect attacks

The tactic “Verify message integrity” refers to the detection of unauthorized modifications of a message and considers data transmission but similar verification for stored data is missing so we added “Verify storage integrity” to indicate the need to define checks to ensure that stored data have not been modified[3]. “Maintain audit trail” was moved here from the branch “Recover from attacks” because an audit trail may allow detection of attacks and even help identifying the attacker. We replaced “Detect intrusion” by “Identify intrusions”, which was split into “by signature” and “by behavior”, the two standard ways to apply intrusion detection.

6.2 Stop or mitigate attacks

To follow standard security terminology [16], a subject is an active entity that can request resources and includes humans and executing processes; thus, we changed the words “user” and “actor” for “subject” in “Authenticate actor” and “Authorize actor” We kept “Identify actors” because although it is not a defense mechanism in itself, identification is needed for authentication, authorization, and audit trail (logging). Authorization is the definition of rules about who can access what and requires an enforcement aspect, which corresponds to the concept of Reference Monitor [16]. We assume here that authorization enforcement is part of Authorize subjects.

As indicated, we eliminated “Limit exposure”, “Limit access”, and “Separate entities” as tactics because these are security principles which define guidelines for designers but have no specific rules to be applied. We added “Manage security information”, which includes the management of keys for cryptography, the secure storage of authorization rules, and other ways to handle security information. The protection of security information is fundamental to have a secure system. “Filter data” is necessary to avoid attacks based on abnormal inputs such as SQL injection, or coming from untrusted sources.

We added “Verify origin of message”, a form of data authenticity verification, necessary to prevent repudiation of actions performed by a subject. The tactic “Establish secure channel” is required before we can hide data content. Once the secure channel is established, message content can be hidden; the tactic “Encrypt data” was renamed as “Hide data”, with the two varieties “Use cryptography” and “Use steganography”; namely, the two basic ways to hide data content.

6.3 React to attacks

The specific functions to react to attacks depend on institutional policies and the type of application, and are performed by the system for all applications or for each specific application, depending on the system’s policies, so we added “Apply institution policies“. The suggested Revoke access and Lock computer are clearly useful in many

[3] This is an important problem in cloud systems.

cases and could be used explicitly (we consider them special cases of "Apply institution polcies"). We renamed "Inform actors" as "Alert subjects", a term usual in security documents.

6.4 Recover from attacks

We moved "Maintain audit trail" from this branch to "detect attacks" because it is related to tracing and identifying attackers. We also added "Audit actions" (to indicate the recovery system task). Similarly to 6.3, we added "Apply institution policies" because different institutions have different ways to recover from attacks. Again, "Restore" could be kept although it is not strictly related to security and should be done by the platform, not by any specific application.

7. Realizing Tactics with Security Patterns

The previous discussion and our past work [15] show that patterns and tactics are complementary rather than alternative concepts. As we proposed in [59], we can use tactics as a step previous to the use of patterns, not an end in itself, selecting the right pattern takes more detailed knowledge, described in all the sections of the pattern. As we discuss in Section 9 when the designer has a significant knowledge of security, using tactics alone may be enough.

Since the notion of tactics does not prescribe any realization, several researchers have proposed their own realizations. Kim at al. [33] reified tactics as reusable UML building blocks that can be plugged in the architecture according to a list of non-functional requirements (NFRs). An approach to make tactics more precise is by formalizing them [3]. Formalizing a vague concept is useful, but not sufficient: there may be many ways to materialize it, and the designer may need to make many assumptions, since real systems design is not a mathematical or formal process but it draws on experience and intuition. Also, most system designers are not experts on security; thus, mere formalization is not always a practical idea. Although specific parts can benefit of formalization, a semi-formal approach seems enough for the larger part of a system; thus, empirical approaches like patterns, which are systematized heuristics, seem a promising direction. Formalization approaches are in line with [44] and rely on plug-ins or templates to apply realizations, whereas security patterns are generic solutions that stipulate conditions to apply as well as consequences. Aspects [44] and S&D patterns [24] are also possible alternatives. Aspects can be based on code or models but start from code implementations where sections are extracted, abstracted, and unified; in other words, they are not strictly architectural constructs. Another alternative is the use of arguments, which starting from tactics lead to patterns, although this approach has only been tried for safety tactics [65]. Our own experience leads us to believe that the most convenient realization of tactics is by the use of security patterns, but more experience is needed to prove this point.

Table 1 shows a correspondence between tactics and security patterns indicating possible realizations of the new tactics. These patterns can be found in [16], except those with explicit references. This table proves an important value of the new tactics,

they have explicit mappings to security patterns, something not possible in the original set. Note also that to build a real system we need to consider all architectural levels; e.g., it is not enough to select Authenticator for example, but we also need to specify its type (password, biometric, etc.), and where it should go in the functional model of the application; existing techniques like [59] and [60] can guide the designer. As mentioned earlier, to make the work of the designer simpler we introduced in [60] the concept of *Security Solution Frames (SSFs)*, which group together related patterns along horizontal and vertical associations for a single high-level policy; for example, a SSF for authentication could include a vertical hierarchy going from Abstract Authentication to Cloud Authentication passing through Distributed Authentication, and associated horizontally to authentication methods such as Passwords or Certificates. Note also that there may be more than one pattern able to realize a tactic. These are indicated by commas, in some cases we need more than one pattern (an "and" of patterns). Again, these ambiguities require further descriptions.

Table 1. Security pattern realizations of tactics

Tactic	**Security pattern realization**
Authenticate subject	Authenticator, credential
Manage security information	Authenticator, Authorizer, Generation and distri-

8. Using the new tactics.

We illustrate the application of the new tactic set with a portion of a model for a financial institution. As described in Figure 6, Customers can be of two types: Owner (principal) or Account User. Note that these are roles and an Owner is a Customer and can also be an Account User. Account Users can perform Transactions of several types on their own accounts. We follow the list of Table 1 to apply systematically all the tactics and their corresponding pattern realizations. We assume that the customers perform remote transactions and that several financial applications share the same platform. While this is a simple example, its purpose is to illustrate the application of tactics; a more complex example would just require more tactics but the selection procedure is the same.

This UML class model is a conceptual functional model which describes the application data, and is commonly used by software designers. We need to start from the semantics of the application. An architectural model is too vague and general to be able to say anything about application security, except for general security measures for types of applications. An architectural unit called "CustomerAccounts", would not allow the application of content-oriented access control (Users can only access their own accounts). Security constraints reflect semantic aspects of the application and extend its conceptual model. After this step, which would be part of the requirements stage, we present a secure architecture.

A designer would need to use her security knowledge and expertise to define or select mechanisms to realize the tactics. A designer using tactics and security patterns gets significant guidance about the selection of security mechanisms so he does not need to know much about security. We show below the systematic application of the four types of tactics. Figure 7 shows the effect of the pattern selection based on these steps.

Detect attacks

Verify message integrity—Digital Signatures with hashing are indicated for this purpose. Since there is only one remote channel it is clear that the communications Customer to Account are the ones to protect in this way. In a more complex system having several channels, without a threat analysis we cannot tell where we should apply this protection. There are also several algorithms for digital signatures, tactics don't help selecting one but the description of this pattern as in [16] would.

Verify storage integrity—We want to make sure that the contents of the information in the persistent storage has not been changed. This action is important to protect against insiders. It can be done using Authentication and Authorization, possibly complemented with Digital Signatures.

Maintain audit trail--We can use here the Security Logger/Auditor pattern [14]. Again, without additional guidance, we would not know that what we want to log are transactions, but the pattern description makes this point clear.

Identify intrusions—Do we use Signature or Behavior-Based IDS? The descriptions in the corresponding patterns [16] can guide the designer.

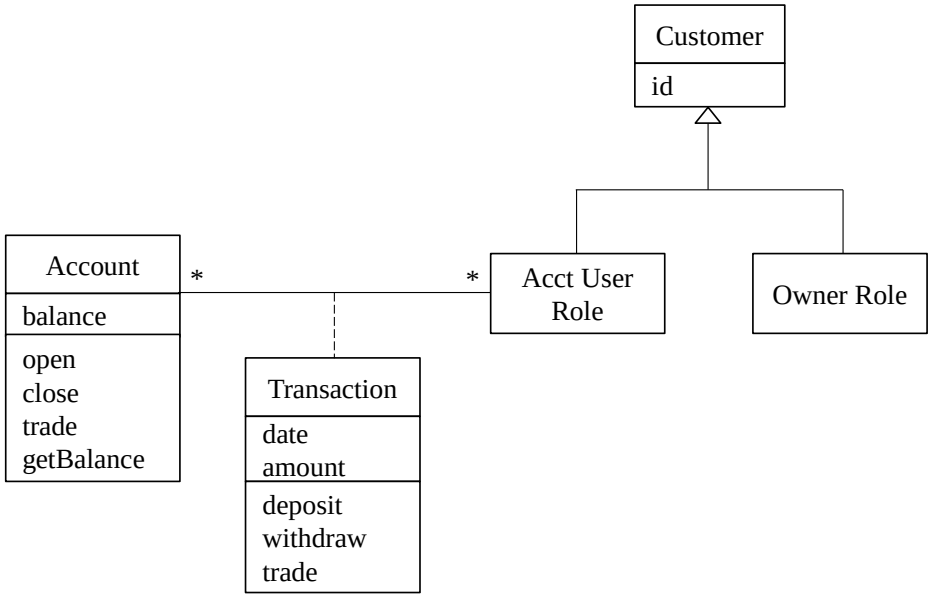

Fig. 6. Part of a conceptual functional model for a financial institution

Stop/mitigate attacks

Authenticate subject—It is clear that the subject is the customer. What is not clear is what type of authentication to use. Again, the pattern text can help. In this case, both Customers and Accounts need to be mutually authenticated as shown by the Authenticator pattern.

Authorize subjects—Needed to access the account and perform transactions. This need comes from threat analysis. Two applications of the Role-Based Access Control pattern control rights for the owner and Account User roles.

Manage security information—Indicates the need to protect the information used for Authentication (identity) and authorization (rights). Only security administrators can access this information to assign rights to users, distribute keys, and similar functions.

Filter data—Use a firewall to filter undesired Internet traffic. The indicated firewall is the Packet Filter Firewall [14].

Verify origin of message—Digital Signatures can identify the origin of customer messages.

Establish secure channel—Use a Secure Channel pattern to protect the channel between Customer and Account.

Hide data—Use Symmetric or Asymmetric Encryption to protect the channel Customer to Account. Selection of type of encryption and specific algorithm depend on the application.

React to attacks

Alert subjects—performed by IDS for all or specific applications

Apply institution policies—We can, for example, cut off traffic from some origin in the presence of an attack from that address.

Recover from attacks

Audit actions—Use log (audit trail) to find the possible perpetrator or determine where to improve security in the system.

Apply institution policies—We should have backup databases and other measures.

As mentioned above, Figure 7 is really a conceptual model, not an architecture. This means we are using tactics already in the requirements stage as conceptual models are the result of that stage. We emphasize that security must start at the requirements stage because it is about semantic constraints, not system structure. Figure 8 shows the result of applying some of the patterns corresponding to the tactics identified above to the corresponding architecture. We have omitted multiplicities for clarity, we also left out the Transaction class and the protection of the security information. The architecture follows an MVC style [8] but there are other architectural styles that can be applied to the architecture for this conceptual model, e.g. an N-tier architecture [16]. The views would be used by customers to access their accounts. Firewalls may have a DMZ structure [16], firewalls may be of several types, and there are two basic types of IDSs. Note that the UML packages indicate patterns which may be instantiated more than once in the model to consider performance or distribution needs. Security Information (SecInfo) is termed Policy Definition Point (PDP) in some models [16]. The model uses Role-Based Access Control for Authorization. The Secure Channel uses Digital Signatures to authenticate messages. The required security mechanisms are not coded but are built using COTS components as in the approaches of [10] and [39].

The point is that tactics alone require the designer to be an expert on security to be able to complete all the design details. On the other hand, using tactics complemented with a catalog of security patterns, a designer who is not a security expert can do a good job to protect the system. Of course, an experienced designer can also use patterns to advantage because of the extra information in them. In this example, we did not use Security Solution Frames [60], which would have made the design even easier for the designer. This example is also a partial validation of our tactics set, we could find easily from Table 1 the patterns we needed to implement the required tactics; using the original set would have been much harder, how do I implement “Limit access” for example?

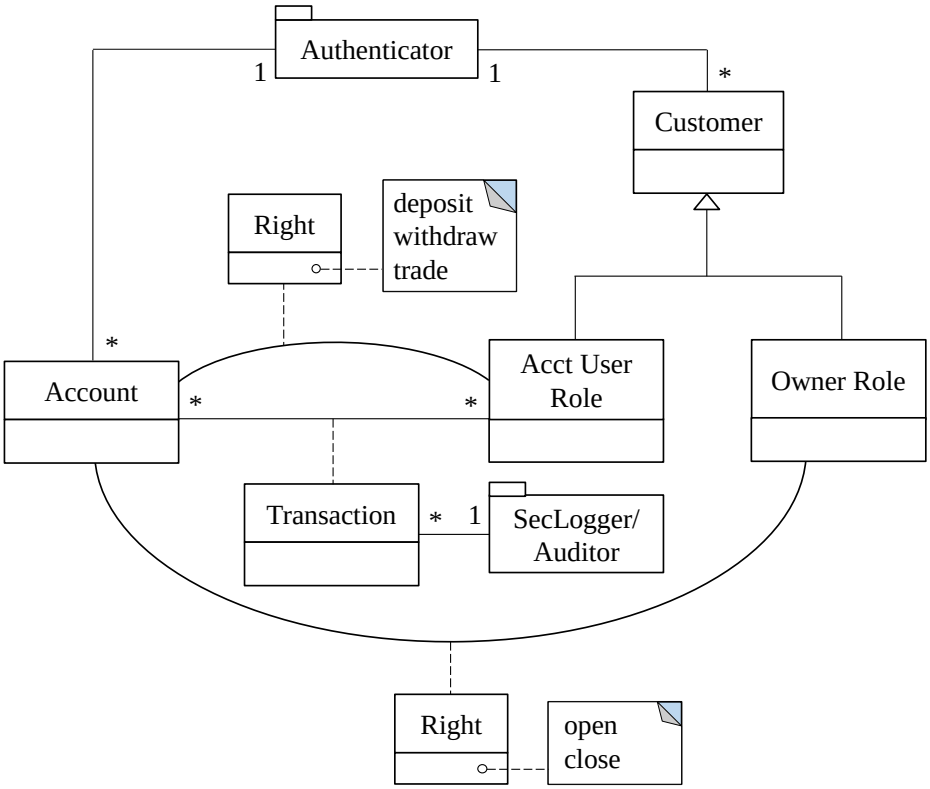

Fig. 7. A partial conceptual model for a secure financial institution

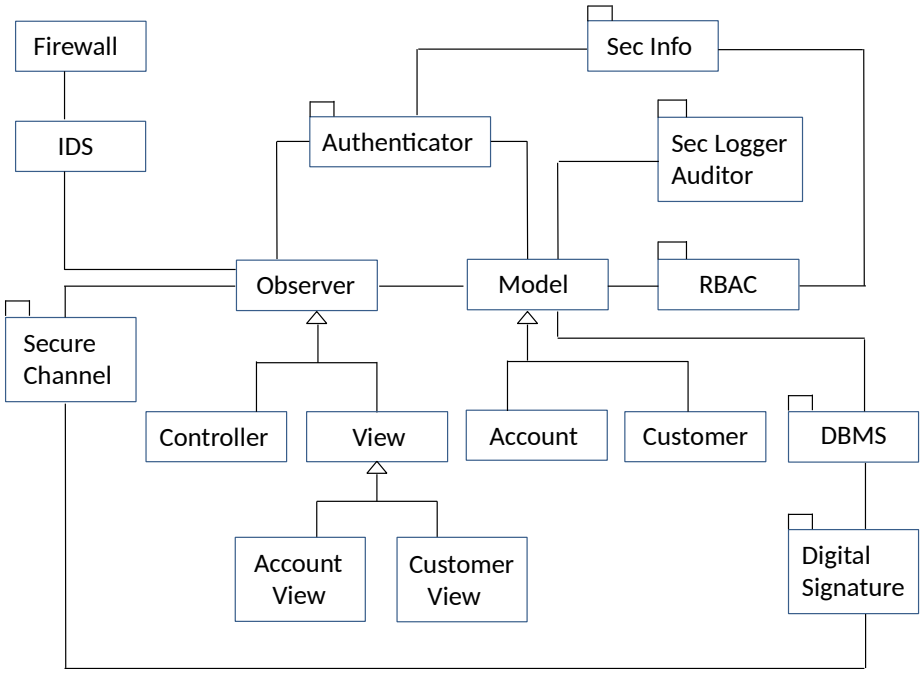

Figure 8. Secure architecture for the conceptual model of Figure 7

9. A Template for Security Tactics

Software architects who are not very experienced and are not used to apply patterns, may be helped by making the tactics somewhat more descriptive. We propose now a template, inspired by the template we use to describe security patterns, in order to guide the designer in selecting the tactics that best respond to a design concern. In particular, this template focuses on providing guidance and support for making decisions to inexpert designers who try to apply security tactics.

The security tactics template includes the following elements:

Name— an active verb that describes precisely and clearly the purpose of the security tactic.

Intent— defines the overall goal that the designer wants to achieve. The goal is closely linked with a stage or the environment where the designer wants to apply the security tactics. For example, if the designer wants to prevent attacks, the environment corresponds to design and the tactics corresponds to the mitigation level. The designer's intention can be:

- *Prevent, avoid or mitigate attacks* so at design time the designer defines actions so that the attack cannot occur or has a reduced effect. Avoidance and mitigation tactics are associated with the implementation of security policies.

- *Detect attacks* at the time they occur or as soon as possible to avoid their impact.

- *React to attacks* that have been discovered using actions to neutralize them for the least possible impact. May imply actions at run time or later.

- *Recovery from attacks* is related to ensuring the availability of the functions and access to software system resources.

Description— briefly summarizes the foundation of the tactic. This summary allows the designer to identify the tactics without elaborating on their operationalization.

Problem— presents in detail a set of security situations that need addressing and motivate the use of the security tactic. Also, this section describes the forces that motivate the application of the security tactic and possible threats and attacks.

Solution — shows how the tactic solves the problems raised in the previous section.

Rationale—indicates the justification, orientations, and reasons behind the solution presented in terms of its advantages and disadvantages.

Realization mechanisms— describe how the designer can implement tactics in the design indicating recommendations and some technical details. Also, it describes common errors and guides the designer to avoid them. For example, security patterns could be used for the purpose expressed in the solution.

Related tactics—are there other known tactics that fulfill a similar purpose or solve the same problems? These can be complementary tactics, variations, or extensions of the described tactic.

Related threats /attacks— allow the association of the current tactic with some common security threats or attacks facilitating the immediate identification of known attacks.

Known uses—examples of common situations where the designer could apply this tactic.

Consequences—indicate the implications of selecting the current security tactic. These consequences define the need to apply other tactics, new situations that must be considered or the restrictions implied when applying this tactic. Also, this section indicates the impact on compliance with the quality attributes. Clearly understanding the consequences of applying tactics allows establishing a roadmap for their introduction into the design.

Table 2 shows the template for the Verify message integrity tactic. This table provides an analysis of the problems giving rise to the use of the tactical solution, proposes two approaches based on the nature of the transmission (local or external), and directs the solution to each particular situation.

Table 2. Template for “Verify message integrity” tactic

Name	Verify message integrity
Intent	Detect attacks that may change a message
Description	A message receiver needs to be sure that the transmitted information has not been changed or deleted in transit
Problem	• The content of a message can be modified or deleted during transmission. • Two varieties of the problem: – Data transmitted on an internal network – Data transmitted on an external network • Attacker origin – internal attacker – external attacker
Solution (Approach)	• Data transmitted on an internal network – Include redundant information in messages • Data transmitted on an external network – Authenticate sender.
Rationale	• Data transmitted on an internal network – Redundant information in messages can detect alterations or loss of contents. • Data transmitted on an external network – A third party can attest about the identity of the message sender or this can be proved by cryptographic means
Realization mechanisms	• Data transmitted on an internal network – Techniques based on maintaining redundant information in messages. ✓Checksums, hash function, cyclic redundancy checks (CRC), parity bit, bit redundancy check digit. • Data transmitted on an external network – Techniques to identify the origin of messages ✓Digital signatures – Message authentication techniques ✓Message Authentication Codes (MAC), Cipher Block Chaining Message Authentication Code (CBC-MAC), Keyed-Hash Message Authentication Code (HMAC).
Related tactics	Establish secure channel
Related threats/attacks	- Trust relationship attacks - Data diddling attacks - Man-in-the-middle attacks
Known uses	- When several organizations need to share information without the existence of a circle of trust.

Consequences	The tactic affects system performance due to the overhead generated by analyzing each message individually.

10. Related Work

As indicated earlier, several authors have tried to relate tactics and security patterns, some of them were discussed in Section 2.

Suntae Kim et al. [33] proposed architectural tactics as reusable UML architectural building blocks that offer generic solutions to specific problems related to quality attributes. Tactics are represented as feature models to support decision making for non-functional requirements through a set of explicit solutions. These solutions are given as UML class and sequence diagrams without the textual sections present in security patterns. In a later paper Kim introduced a quantitative approach to select optimal building blocks [34]. However, this approach is very rigid and does not give the designer the freedom provided by patterns. The choice of blocks is also limited, since no extensive catalog of these building blocks exists. A problem for optimal selection is that quantitative measures for security are very hard to define and no measure is acceptable for all applications [52]. A good point in this approach is the ability to handle other types of tactics, such as reliability and performance.

Ray et al. [44] proposed tactics as an intermediate architectural concept between high-level decisions and architecture patterns; that is, in their view architecture patterns directly implement architectural tactics, but as we have shown, this mapping is far from simple.

Ryoo et al. [47, 48] defined a methodology to extract tactics from security patterns through activities such as reclassification of architectural patterns, decomposition of patterns, derivation of tactics hierarchies applying reasoning and intuition over patterns, and realization or instantiation of existing tactics. They evaluated several sets of security architectural patterns and applied a Delphi technique to yield a new security tactics hierarchy, shown in Figure 9. It includes tactics “Limit exposure” and “Limit access”, which we argued for removal, and redundancies like “Maintain confidentiality” (shown as a separate tactic, but which requires Authentication and Authorization, which are also tactics). Similar arguments to the ones we made earlier would apply to their classification.

Ryoo et al. [49] studied the use of security tactics in 53 open source software projects. They arrived to the conclusion that several tactics were rarely used or not use at all, which confirms our elimination of those tactics.

Dae-Kyoo Kim et al. considered using patterns to let architectures adapt to changes in non-functional requirements, which they call “architectural sustainability” [32]. Wu and Kelly produced a template for safety tactics which has several aspects in common with ours [65], which is not surprising since safety and security have several common

aspects. Rehn [45] discussed the advantages of tactics for security and safety, showing how they can be applied using patterns.

[1] studies mappings between security tactics and patterns to see how easily a tactic can be implemented in an architectural pattern. They work with 8 tactics of the original set and see, for example, that the shared directory, microkernel, and MVC can easily implement these tactics, while the Interpreter can only easily implement the Audit Trail pattern. Their use of tactics is similar to [23], where tactics are added to architectural units.

In addition to secure or reliable design, tactics also have value for other purposes. Cañete [9] used tactics to annotate Jackson's problem frames; the annotations provide arguments for satisfaction of quality factors, which may include security. As mentioned in Section 2, Harrison and Avgeriou [27] used tactics for annotations in architectural patterns about design decisions, which may be more general than security.

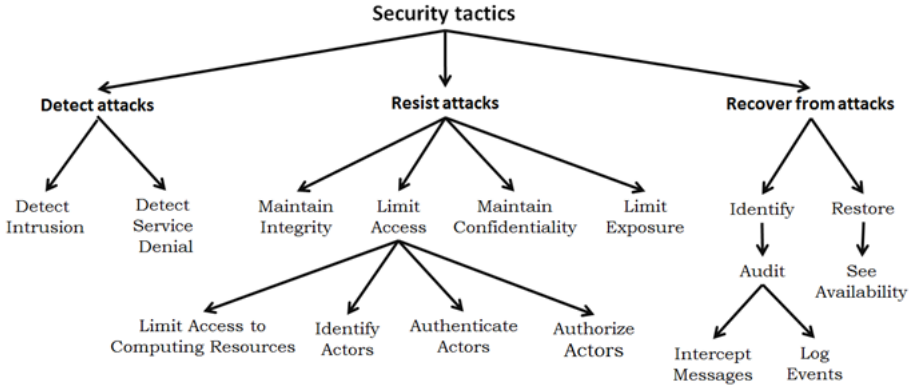

Figure 9. Classification of security tactics according to [20]

11. Validation

How should a specific set of tactics be validated? The original proposal [4] did not explicitly validate the initial set of tactics (at least we have found no validation), and indeed no formal or argued parsimony and completeness arguments have been given; conversely, using support from a Delphi approach as in [48] seems too weak a rationale for this purpose because of the subjective nature of this approach; it is not clear either if the participants were security experts. The only experimental work about the use of tactics in industry is the work of Ryoo et al. [49], which as indicated in Section 10 showed that some tactics are rarely used or not used at all; this work justifies our elimination of some tactics but not our new tactics

Since tactics are only indications of what controls we need to make a system secure and they don't define a complete methodology, it is not possible to validate them experimentally. The architects using them may have more or less experience and knowledge of security and it would not be clear what to measure. If we consider a methodology using a combination of tactics and patterns, as in the latest version of our methodology [59], it would be possible to perform an experimental evaluation; however, this study would not really measure the value of our new set of tactics because we cannot separate its effect from the whole methodology. We prefer then perform a qualitative evaluation where we consider factors such as completeness, ease of use, and how well they cover the protection of the system as a whole. To validate our new set we use the approach of Hug et al. [29], for which we showed the application of the new tactics and their corresponding patterns to a real example (see Section 8). While this example is small, the approach is compositional in that tactics (and patterns) can be applied incrementally to each module of a complex system. We also checked to see if our set covers all the units of a system according to different architectural models. Finally, we looked at all the known secure software development methodologies and textbooks to verify that this coverage.
However, an application of our tactics to a specific system cannot prove that our tactics are better than other proposed sets unless we have an accepted way of measuring the degree of security reached with each set of tactics, but no such measure exists. In the case of using the tactics as part of a pattern-based methodology, the results are even fuzzier, it would be very hard to separate the effect of the tactics from the other stages of the methodology

Table 1 and its use in the example of Section 8 demonstrate completeness and ease of use by showing that the new tactics have a direct and clear realization using security patterns, something that is unclear how to do in the original set. It is clear that a security methodology tries to produce a secure system and thus it should cover all weak points.

Another aspect of validation is to show that the new set covers all assets that need to be protected in a system, as shown in Figure 10 which is a standard decomposition of an architecture using hierarchical layers. The use of operations in the application can be controlled by Authentication and Authorization, and similarly access to data in the database system. Data in an application could be sent out to another application through a channel; this can be protected by encryption. The database usually includes the authorization rules and related system procedures, which are also protected by protecting the database system. As further verification, we can point that in [59] we used a different system decomposition, along functional instead of architectural layers, and arrived to a similar tactic set. In this decomposition we can see that the complete functional decomposition for distributed systems can be protected with the proposed tactics:

User interaction—Protected with tactics Authenticate subject, Authorize subject, and Filter data

Data/storage management—Protected with tactics Authenticate subject, Authorize subject, Manage security information, Hide data, Maintain audit trail, and Verify storage integrity

Resource management—Protected with tactics Authenticate subject, and Authorize subject.

Distribution control-- Protected with tactics Authenticate subject, Authorize subject, and Identify intrusions

Communication-- Protected with tactics Verify message integrity, Identify intrusions, Filter data, and Establish secure channel

Addressing-- Protected with tactics Identify intrusions, and verify message integrity.

We can consider even another architectural decomposition, used in many architectures: the Model View Controller architecture pattern, used in our example of Figure 8 [57]. In this case we found the need to add to the basic MVC architecture: an Authenticator to assure that messages are legitimate; an RBAC Authorizer, to control access to the data in the model, a Secure Channel because the views and controllers can be remote, a Security Logger/Auditor to record use of the data in the model, and a Filter (Data Sanitizer) to prevent spurious commands [16]. All these functions are available as patterns in the new set of tactics.

We searched the known secure development methodologies [58] and examined their artifacts and steps. As a final check we consulted several standard security textbooks, including [2, 25, 42]. Their discussions confirmed our assumptions. Requirement studies such as [11] and [36] also confirmed our choice. We can note that Wu and Kelly [65] found 17 safety tactics based on a literature survey and not by experiment.

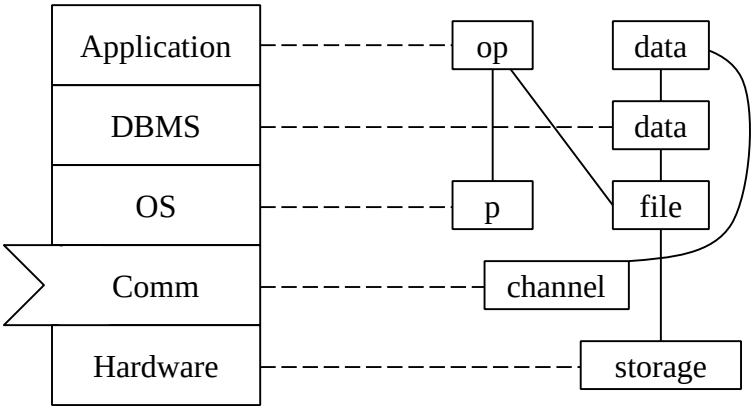

Fig. 10. Assets that must be protected (op=operation, p=process) by security tactics

12. Conclusions

Building secure systems demands an approach where security can be specified from the start of software development, considering the semantics of the application, and iterating between requirements and design. We have studied how tactics and security patterns can be used in this sequence. We started by revisiting existing security tactics and a modified set of tactics was derived by combining architectural and security knowledge; we do not claim they are complete or optimal, but we believe that they are more useful (easier to apply, more efficient) than previous sets. In particular, unlike previous tactic sets, these can be directly realized with existing security patterns, which is a clear advantage. We showed a possible realization in terms of security patterns and provided an example. By connecting requirements, software architecture, and implementation aspects we have contributed to the communication between communities which usually work independently of each other but whose joint contributions are needed to produce secure systems. For those who prefer to use only tactics we proposed a template to make them easier to use by those with less experience.

We have not intended to denigrate the original set of tactics, which we consider a pioneering effort to add security to software architectures, but we have tried to make that set more efficient in protecting systems.

We insist that the new set of tactics is a refinement of the initial set, not a replacement. We also addressed some confusion in terminology, definitions, and relationships among security patterns and tactics, which has led to methodologies that are difficult to combine with each other and with unclear impact on security. In time, precise definition of these concepts should lead to better architectural knowledge and better methodologies to build secure systems.

Tactics are important because they conceptualize what is needed, what specific aspects of the systems must be secured. This seems to be their main value and our approach is based on this idea. For experienced architects with a good knowledge of security, these security 'hints" may be sufficient; however, for most architects and developers we need to provide further guidance, this is where our extended tactics and patterns are valuable.

The use of tactics combined with our secure methodology produces a systematic hardening of a system, where functional and non-functional aspects are developed together. We need to produce more examples of the application of this methodology to further validate this set of tactics and to convince industry of the value of a systematic way as the only road to build secure software.

Acknowledgements

This work was started by the visit of Dr. Fernandez to Chile by invitation of CONICYT (Chilean National Science Foundation) in their program of foreign visitors. It was continued through partial support from CONICYT through grants FONDECYT 1140408 and CCTVal FB0821. This paper is an extended version of

our paper in ECSA 2015. We thank the anonymous referees of ECSA for their comments that improved this paper. We also thank Dr. Anton Uzunov for valuable discussions.

References

1. A Alebrahim, S Fassbender, M Filipczyk, M Goedicke, M Heisel. "Towards a reliable mapping between performance and security tactics, and architectural patterns", *Proceedings of the 20th European Conference on Pattern Languages of Programs*. Irsee 2015.
2. R. Anderson, Security Engineering (2nd. Ed.), Wiley, 2008.
3. H. Bagheri and K. Sullivan, "A formal approach for incorporating architectural tactics into the software architecture", *Procs. of SEKE 2011*, 770-775.
4. L. Bass,M. Klein, and F. Bachmann, "Quality attribute design primitives and the Attribute Design method", SEI, Carnegie Mellon University, 2001.
5. L. Bass, P. Clements, and R. Kazman, *Software architecture in practice* (2nd Ed), Addison-Wesley 2003.
6. L. Bass, P. Clements, and R. Kazman, *Software architecture in practice* (3rd Ed), Addison-Wesley 2012.
7. A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software", Chapter 16 in Pattern Languages of Program Design 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.).
8. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal, *Pattern-oriented Software Architecture*, Wiley, 1996.
9. J. M. Cañete, "Annotating problem diagrams with architectural tactics for reasoning on quality requirements", *Information Proc. Letters*, 112, 2012, 656-661.
10. H. Cervantes, R. Kazman, J. Ryoo, D. Choi, and D. Jang. Architectural approaches to security: Four case studies. IEEE Computer, 49:60–67, 2016.
11. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *NFRs in software engineering*, Kluwer Acad. Publ., Boston, 2000.
12. R. J. Ellison, A.P. Moore, L. Bass, M. Klein, F. Bachmann, "Security and Survivability: Reasoning Frameworks and Architectural Design Tactics", Technical Note CMU/SEI-2004-TN-022, September 2004
13. 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.
14. 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.
15. E. B. Fernandez, and H. Astudillo, "Should we use tactics or patterns to build secure systems?", *First International Symposium on Software Architecture and Patterns*, in conjunction with the 10th Latin American and Caribbean Conference for Engineering and Technology, July 2012, Panama City, Panama, 23-27.
16. E. B. Fernandez, *Security patterns in practice - Designing Secure Architectures Using Software Patterns*, Wiley Series on Software Design Patterns, June 2013.
17. E. B. Fernandez, Nobukazu Yoshioka, Hironori Washizaki, and Joseph Yoder, "Abstract security patterns for requirements specification and analysis of secure systems", *Procs. of the*

WER 2014 conference, a track of the 17th Ibero-American Conf. on Soft. Eng.(CIbSE 2014), Pucon, Chile, April 2014.

18. E. B. Fernandez, R. Monge, R. Carvajal, O Encina, J. Hernandez, and P. Silva, R."Patterns for Content-Dependent and Context-Enhanced Authorization". *Proceedings of 19th European Conference on Pattern Languages of Programs*, Germany, July 2014.
19. E.B.Fernandez, H. Astudillo, and G. Pedraza-Garcia, "Revisiting architectural tactics for security", *9th European Conf. on Software Architecture (ECSA 2015),* September 5-7, 2015.
20. E.B.Fernandez, N.H. Washizaki, "Evaluating the degree of security of a system built using security patterns", submitted for publication. Available from the first author.
21. D.G.Firesmith, "Engineering safety and security-related requirements for software-intensive systems", Tutorial, 32nd Int. Conf. on Soft. Eng., May 2010.
22. R.Flanders and E.B.Fernandez, "Data filter architecture pattern using distributed filter components", *Proceedings of the Conference on Pattern Language of Programs (PloP'99*).
23. Olga Gadyatskaya; Fabio Massacci; Yury Zhauniarovich, "Security in the Firefox OS and Tizen Mobile Platforms", Computer , 2014, Volume: 47, Issue: 6 , 57 - 63, DOI: 10.1109/MC.2014.165
24. B. Gallego, A. Muñoz, A. Maña, D. Serrano, "Security patterns, towards a further level", *Procs. SECRYPT 2009*, 349-356.
25. D. Gollmann, *Computer security* (3nd Ed.), Wiley, 2011.
26. M. Hafiz and R.E.Johnson, "Evolution of the MTA architecture: the impact of security", *Software—Practice and Experience*, vol. 38, 2008, 1569-1599, doi: 10.1002/spe.880
27. N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation", *The Journal of Systems and Software*, 83, 2010, 1735-1758.
28. HIPAA "Public Law 104-191: Health Insurance Portability and Accountability Act of 1996", August 21, 1996, 104th U.S. Congress, Washington D.C. http://www.hipaa.org/
29. Charlotte Hug , Agnès Front, Dominique Rieu, Brian Henderson-Sellers, A method to build information systems engineering process metamodels, *The Journal of Systems and Software*, Vol. 82, 10, Oct. 2009, 1730-1742.
30. IBM Corp., "Introduction to business security patterns", white paper, http://www-03.ibm.com/security/patterns/intro.pdf (last accessed July 27, 2016).
31. A. Kavanagh, "OpenStack as the API framework for NFV: the benefits, and the extensions needed", Ericsson Review, 2015, No 3, 2-7.
32. D.K.Kim, J. Ryoo, S. Kim, "Building sustainable software by preemptive architectural design using tactic-equipped patterns", *Procs. 9th Int. Conf. on Availability, Reliability, and Security (ARES)*, 2009, 484-489.
33. S. Kim, D.-K. Kim, L. Lu, S. Park, "Quality-driven architecture development using architectural tactics", *The Journal of Systems and Software*, 82 (8), 2009, 1211-1231.
34. S. Kim, "A quantitative and knowledge-based approach to choosing security architectural tactics", *Int.J. Ad Hoc and Ubiquitous Computing*, vol. 18, Nos. 1/2, 2015, 45-53.
35. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood, "seL4: Formal verification of an OS kernel". *22nd ACM Symposium on Operating System Principles*, October 2009.
36. Fabio Massacci, Federica Paci, Le Minh Sang Tran, Alessandra Tedeschi: Assessing a requirements evolution approach: Empirical studies in the air traffic management domain. Journal of Systems and Software 95: 70-88 (2014)
37. D Mellado, E Fernández-Medina, M Piattini, "A comparison of software design security metrics", Proceedings of the Fourth European Conference on Software Architecture, 2010, 236-242

38. Microsoft, Web services security patterns, https://msdn.microsoft.com/en-us/library/ff648183.aspx , (accessed July27, 2016)
39. S. Moral-Garcia, S. Moral-Rubio, E. B. Fernandez, and E. Fernandez-Medina "Enterprise Security Pattern: A Model-Driven Architecture Instance", *Journal of Computer Standards & Interfaces (*Special Issue on Security in Information Systems), vol. 36, No 4,748-758, 2014.
40. P. G. Neumann, *Principled assuredly trustworthy composable architectures*, Final SRI report to DARPA, December 28, 2004.
1. Juan C. Pelaez, “Using Misuse Patterns for VoIP Steganalysis”, DEXA Workshops 2009: 160-164, 2008
2. C.P.Pfleeger, *Security in computing, 6th Ed*., Prentice-Hall, 2015.
41. C. Preschern, "Catalog of Security Tactics linked to Common Criteria Requirements", *Procs. of PLoP 2012*.
42. I. Ray, R. B. France, N. Li, G. Georg, “An aspect-based approach to modeling access control concerns”. *Inf. & Soft. Technology*, (9): 575-587 (2004).
43. Christian Rehn, “Software Architectural Tactics and Patterns for Safety and Security”, www.christian-rehn.de/downloads/seminar_safe_sec.pdf (last accessed August 20, 2016).
44. N. Rozanski, E. Woods, “Software systems architecture: working with stakeholders using viewpoints and perspectives”. Addison-Wesley Educational Publishers., 2nd Ed. (2012)
45. J. Ryoo, P. Laplante, and R. Kazman, “A methodology for mining security tactics from security patterns”, *Procs. of the 43rd Hawaii International Conference on System Sciences*, 2010, http://doi.ieeecomputersociety.org/10.1109/HICSS.2010.18
46. J. Ryoo, P. Laplante, and R. Kazman, “Revising a security tactics hierarchy through decomposition, reclassification, and derivation”, *2012 IEEE Int. Conf. on Software Security and Reliability Companion*, 85-91.
47. J. Ryoo, B. Malone, P. A. Laplante, P. Anand, “The use of security tactics in open source software projects”, *IEEE Trans. On Reliability, vol. 65, No 3*, September 2016, 1195-1204.
48. J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems”, *Procs. of the IEEE*, vol. 63, No 9, Sept.1975, 1278-1308.
49. SANS, “Information security policy templates”, http://www.sans.org/security-resources/policies/
50. Reijo M. Savola: “Quality of security metrics and measurements”. *Computers & Security* 37: 78-90 (2013)
51. J. S. Shapiro, N. Hardy, “EROS: A Principle-Driven Operating System from the Ground Up”, *IEEE Software*, Jan/Feb 2002.
52. D. Schmidt, P. Stephenson, “Experience using design patterns to evolve communications soft
are across diverse OS platforms”, *Proceedings of the 9th European Conference on Object-*
Oriented Programming, Aarhus, Denmark, August 7-11, 1995.
53. C. Steel, R. Nagappan, and R. Lai, Core Security Patterns: Best Strategies for J2EE, Web Services, and Identity Management, Prentice Hall, Upper Saddle River, New Jersey, 2005.
54. Marianne Swanson and Barbara Guttman, “Generally Accepted Principles and Practices For Securing Information Technology Systems”, NIST Special Publication 800-14, U.S. DEPARTMENT OF COMMERCE, NIST, September 1996
55. R. N. Taylor, N. Medvidovic, and N. Dashofy. *Software Architecture: Foundation, Theory, and Practice*, Wiley, 2010.
56. 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, 2920-3006.

57. A.V. Uzunov, E. B Fernandez, Katrina Falkner, “ASE: A Comprehensive Pattern-Driven Security Methodology for Distributed Systems”, *Journal of Computer Standards & Interfaces,* 2015, http://dx.doi.org/10.1016/j.csi.2015.02.011
58. A.V. Uzunov, E. B Fernandez, Katrina Falkner, “Security solution frames and security patterns for authorization in distributed, collaborative systems”, *Computers & Security*, 55, 2015, pp. 193-234, doi: 10.1016/j.cose.2015.08.003
59. M. VanHilst, E. B. Fernandez, and F. Braz, "A multidimensional classification for users of security patterns", *Journal of Res. and Practice in Information Technology*, vol. 41, No 2, May 2009, 87-97.
60. H. Washizaki, E.B.Fernandez, K. Maruyama, A.Kubo, N. Yoshioka, “Improving the classification of security patterns”. *Procs. 20th International Workshop on Database and Expert* Systems Application (DEXA '09), Aug. 31-Sept. 4, 2009
61. William G. Wood, “A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0, February 2007, TECHNICAL REPORT CMU/SEI-2007-TR-005ESC-TR-2007-005
62. E. Woods, N. Rozanski, “Using architectural perspectives”, *Procs. of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA’05))*.
63. W. Wu, T. Kelly, “Safety tactics for software architecture design”, *28th Ann. Int. Conf. of Computer Software and Applications*, 2004, 368-375.