

Usability of Security Specification Approaches for UML Design: A Survey *

C. Talhi, D. Mouheb, V. Lima, M. Debbabi and L. Wang

Computer Security Laboratory, Concordia Institute for
Information Systems Engineering, Concordia University

M. Pourzandi

Software Research, Ericsson Canada, Town of Mount-Royal, Canada

Since it is the de facto language for software specification and design, UML is the target language used by almost all state of the art contributions handling security at specification and design level. However, these contributions differ in the covered security requirements, specification approaches, verification tools, etc. This paper investigates the main approaches adopted for specifying and enforcing security at UML design and surveys the related state of the art. The main contribution of this paper is a discussion of these approaches from usability viewpoint. A set of criteria has been defined and used in this usability discussion. The discussed UML approaches are stereotypes and tagged values, OCL, and behavior diagrams. Extending the UML meta-language or creating new meta-languages for security specification are also covered by this study.

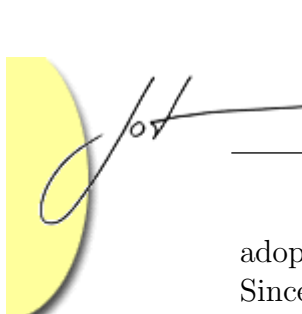
1 INTRODUCTION

Security is a challenging task in software engineering. Indeed, security has been widely investigated by the software engineering community during the last decades. This covers specifying security requirements and enforcing them on software. Software security enforcement is generally conducted as an afterthought phase of the software development life cycle. However, this practice is no longer acceptable for such an important aspect, especially with the increasing complexity and pervasiveness of today's software systems. Since it is the de facto language for software specification and design, the Unified Modeling Language (UML) [12] is the target language used by almost all state of the art contributions handling security at specification and design level.

Many contributions have been presented in the state of the art for specifying and enforcing security at UML design [1, 2, 4, 5, 6, 7, 8, 14, 16, 17, 18, 19, 20, 22, 23, 27, 28, 30]. While sharing almost the same objectives, these contributions

Cite this document as follows: http://www.jot.fm/general/JOT_template.LaTeX.tgz

* The research leading to this work was possible due to funding and scientific collaboration with Software Research, Ericsson Canada.



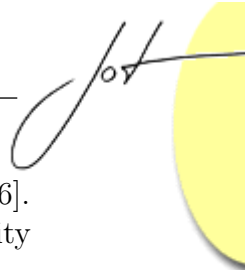
adopt different approaches for security requirements specification and enforcement. Since there is no consensus nor standard on how security should be specified for UML design, non-security experts designers are feeling lost when it comes to deal with security aspects of their design. In fact they are looking for precise answers to many questions where the main important ones are: (1) What are the main approaches that can be adopted for security specification? (2) How each approach can be used for security specification? (3) For a given security requirement, what are the possible specification approaches and if possible what is the best one (if any)? and (4) What are the limitations of each approach in terms of tool support and complexity? Unfortunately, as far as we know, the state of the art is not providing such precise answers. In fact we did not find any contribution covering all these aspects in the same study providing UML designers with the expected answers.

This paper tries to answer the aforementioned questions by (1) surveying the state of the art related to UML security specification and identifying the main adopted approaches, (2) explaining how each approach can be used for security specification, and (3) defining a set of usability criteria and using them to discuss the usability of each approach. The set of usability criteria was inspired from the state of the art of software usability and software security requirements specification. From studying the state of the art, we identified three main approaches that have been followed for UML security specification. The first approach is based on using the language artifacts provided by standard UML: Stereotypes, the Object Constraint Language (OCL) [11], and behavior diagrams. In the second approach, the UML meta-language is augmented by new language constructs allowing the specification of security requirements. The third approach consists in defining a new specification language to specify security requirements on UML diagrams. These approaches will be presented and their usability will be discussed in this paper. In the sequel, first, we briefly present the main classification of security requirements covered by the study. Then we discuss the related work.

Security Requirements

For reasons of space and time, in the following we do not discuss individual security requirement. Instead, we classify security requirements and discuss how different UML approaches can represent them. Numerous classifications can be found in the literature. The main classifications considered in this paper are the following:

Static vs Dynamic Enforceability Classification: Some security requirements are classified as *statically-enforceable*, which means there exists some algorithm that, when applied on a static representation of the application (design, source code, abstract representation, etc.), can decide in a finite period of time whether the application satisfies or not the security requirement [26]. Examples of these security requirements are secrecy, integrity, authenticity, etc. When such an algorithm cannot be defined, a security mechanism should be designed to be deployed as a controller of the application. It will be executed in parallel to the later and in-



tervene whenever the application is about to violate the enforced requirement [26]. Such security requirements are classified as *dynamically-enforceable*. To this security requirements class belong authorization, non-repudiation, and privacy.

Logic Classification: Different logics have been used to formally specify security requirements. Usually, a security policy is specified using some logic and then a verification tool (a model checker or a theorem prover) is used to check whether the design satisfies or not that security policy. Thus we limit the scope of logic classes studied in this paper to the main logics used by verification tools. These logic classes are mainly the Linear Temporal Logic and the Branching Temporal Logic[21]. Linear Temporal Logic includes languages such as LTL, PLTL, etc. Security properties that belong to this logic such as secrecy, integrity, authentication, fair exchange, non-repudiation, etc, are expressed based on a linear sequence of states that represent the system execution events. Branching Temporal Logic such as CTL, CTL* is based on a branching notion of time, which means the system is viewed as a tree of states representing all the possible execution paths rather than a linear sequence of states. To this security requirements class belong secrecy, integrity, and authenticity.

Related Work

To the best of our knowledge, this is the first work studying the usability of the main approaches adopted for specifying and enforcing security requirements at UML design. We found in the literature few contributions that discuss and evaluate the UML language, UML stereotypes and OCL. However, none of them addresses the usability of these approaches.

Gogolla and Henderson-Sellers [9] provide an analysis of UML stereotypes and propose some suggestions to improve the definition and use of stereotypes within the UML meta-model. They use OCL to define precise stereotypes, and suggest that the UML meta-model should be adjusted and tool support should be provided to deal with stereotypes.

Schleicher and Westfechtel [25] discuss and evaluate the UML meta-language. A classification of stereotypes and a comparison of different approaches of extending the UML is also given. Finally, the paper proposes various ways to extend the UML meta-model for better readability, expressiveness, and verifiability of the extensions.

Regarding OCL, [13] discusses a number of issues related to the syntax and semantics of OCL such as navigation, state models, object creation, etc. In addition, the paper proposes some solutions for clarification and extension of the OCL.

The remainder of this paper is organized as follows. Section 2 surveys the state of the art related to specifying security requirements for UML design and classifies the existing approaches. Section 3 starts by defining a set of usability criteria and then uses them to discuss the usability of each of the security specification approaches presented in Section 2. Finally, Section 4 concludes the paper with some summarizing observations.

2 SPECIFYING SECURITY REQUIREMENTS FOR UML DESIGN

In this section we investigate security specification for UML design. We start with surveying the state of the art. According to our survey, there are three main approaches that are usually adopted: using UML artifacts, extending the UML metalanguage, and creating a new metalanguage. Thus, a subsection is dedicated to present each approach and show how it can be used for security specification.

State of the Art Survey

In this section, we present a survey on the main state of the art contributions that are related to specifying and designing security for UML.

The UMLSec approach by Jürjens is among the first efforts in extending UML for the development of security-critical systems [14]. It provides a UML profile where general security requirements such as secrecy, integrity, fair exchange, etc are encapsulated using UML stereotypes and tagged values. It also defines a tailored formal semantics to formally evaluate UML diagrams against weaknesses. In order to analyze security specifications, the behaviour of a potential adversary that can attack various parts of a system is formally modeled. However, UMLSec lacks in expressiveness since security properties are predefined using UML stereotypes and tagged values. This framework cannot be used to specify user-defined properties.

Pavlich-Mariscal *et al.* propose an aspect-oriented approach to model access control policies [20]. They augment UML with new diagrams to represent Role-Based Access Control (RBAC), Mandatory Access Control (MAC) and Discretionary Access Control (DAC) schemes. that are separated from the main design. MAC, DAC and RBAC are decomposed into security features which represent specific elements of an access control policy, e.g. permissions, MAC security properties, delegation rules, etc. This is the only approach that combines MAC, DAC and RBAC into a set of security diagrams separated from the main design. Modeling security as aspects reduces the scattering of access control definitions in the entire application. It is also possible to make changes to the design without impacting the entire security of the application. Moreover, Pavlich *et al.* supports an AOP [15] code generation to enforce access control policies at execution time. However, this approach is limited to access control policies.

Zisman proposes a framework to support the design and verification of secure peer-to-peer applications [30]. The design models and security requirements are specified using UMLSec. The modeling of abuse cases to represent possible attack scenarios and potential threats helps designers to identify the security properties to be verified in the system. In addition, this approach artifacts expressing properties to be verified by defining a graphical template language. It also allows verification of the models against the properties and visualization of the verification results.

Lodderstedt *et al.* (SecureUML) propose an approach to model RBAC poli-



cies for model-driven systems [17]. It also provides additional support to specify authorization constraints related to the state of the system. In contrast to other approaches, SecureUML proposes a general schema for building systems by combining design modeling languages with a security modeling language; it does not fix one particular design modeling language. However, it only focuses on specifying RBAC model, and does not support secure code generation.

The approach of Doan *et al.* incorporates RBAC, MAC and lifetimes into UML for time-sensitive application design [6]. The main focus of this approach is that the process of designing and integrating security in a software application captures not only the current design state, but allows tracking the entire design evolution process via the creation and maintenance of a set of design instances over time. The design tracking allows a software/security engineer to recover to an earlier design version that satisfies specific security constraints.

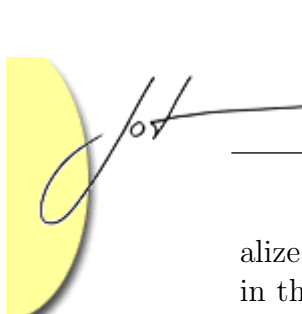
Montangero *et al.* (For-LySa, DEGAS project) present two UML profiles to model authentication protocols [18]: the *Static For-LySa profile* which describes how the authentication protocol concepts (Server, Principals, Keys, Messages, etc.) can be modeled using UML class diagrams, and the *For-LySa profile* which models the dynamic aspects of the protocol in sequence diagrams, as well as the information needed to analyze the protocol. In order to validate a protocol, the approach For-LySa defines a specification language with semantics to write pre/post conditions and invariant constraints. This approach focuses only on the modeling of authentication protocols.

The approach of Ray *et al.* uses parameterized UML diagrams to model RBAC and MAC frameworks and then compose them manually to produce a hybrid access control policy [23]. It is the first approach that attempts to combine RBAC and MAC. However, it focuses only on how to model RBAC and MAC systems in UML without considering how this approach can be used to design a secure software system. In another effort [27], Ray *et al.* integrate RBAC and MAC policies into an application using an aspect-oriented approach to separate access control features from other application features.

Alghathbar and Wijesekera (AuthUML) propose a framework to incorporate access control policies into use case diagrams only [2]. The aim of AuthUML is analyzing (not necessarily modeling) access control policies during the early stages of the software development life cycle before proceeding to the design modeling to ensure consistent, conflict-free and complete requirements.

Popp *et al.* propose an extension to the conventional process of developing use case oriented processes [22]. In addition to modeling security properties with UML, this approach provides a method to incorporate these security aspects into a use case oriented development process.

Painchaud *et al.* (SOCLe project) provide a framework that integrates security into the design of software applications [19]. It also includes verification of UML specifications and a graphical user interface tool that allows the designer to visu-



alize the verification results and to inspecte the diagrams' execution graph. But in this approach, security policies are simply specified using the Object Constraint Language (OCL) constraints.

Ledru *et al.* (EDEMOI project) aim at modeling and analyzing airport security [16]. The security properties are first extracted from natural language standards and documents, and integrated into UML diagrams as stereotypes in a UML profile. The UML specifications are then translated into formal models for verification purposes. This approach is not general enough to be used for software development.

Epstein and Sandhu's work is one of the first approaches that investigate the use of UML to model RBAC policies [7]. However, it is limited to only one specific RBAC model which is the RBAC Framework for Network Enterprises (FNE). The FNE model contains seven abstract layers that are divided in two different groups. This approach allows to present each of the FNE model's layers using UML notation by defining new stereotypes. This approach can assist the role engineering process, however, it does not include subtle properties of RBAC such as separation of duty constraints and it does not provide a method for deriving roles. In addition, there is no formal semantics for verifying UML models.

Ahn and Shin propose a technique to describe the RBAC model with three views: static view, functional view and dynamic view using the UML diagrams [1]. This approach focuses only on the way that UML elements can be used to model RBAC policies rather than taking a larger view of examining secure software design. It does not provide a systematic modeling approach that can be used by developers to create applications with RBAC models.

Brose *et al.* extend UML models to support the automatic generation of access control policies for CORBA-based systems [4]. They specify both permissions and prohibitions on accessing system's objects since the analysis phase in use case diagrams. The UML design is used to generate an access control policy in VPL (View Policy Language) that is deployed together with the CORBA application.

Vivas *et al.* propose an approach for the development of business process-driven systems where security requirements are integrated into the business model [28]. Security requirements are first stated at the high level of abstraction within a functional representation of the system given by UML diagrams using tagged values. Next, the UML specification is translated into XMI representation that allows automatic processing of the specification. Finally, the resulting specification is translated into a formal notation for consistency checking, verification, validation and simulation.

Fernandez provides a methodology to build secure systems using patterns [8]. The main idea of this approach is that security principles should be applied through the use of security patterns at every stage of the software development process (requirements, analysis, design and implementation stages). At the end of each stage, audits are performed to verify that the security policies are being followed.

Chan and Kwok [5] propose a design methodology for e-commerce systems to specify design details for three processes: Risk, Engineering, and Assurance, which



Table 1: The Use of Security Specification Approaches in the State of the Art.

Contributions	Stereotypes and tagged values	OCL	Behavior diagrams	Extending the UML metalanguage	New metalanguage
UMLSec [14]	✓				
P. Mariscal <i>et al.</i> [20]	✓			✓	
SecureUML [17]	✓	✓			✓
Zisman [30]			✓		
SOCLe [19]		✓			
For-LySa [18]	✓				
Epstein and Sandhu [7]	✓				
Brose <i>et al.</i> [4]	✓				
Ahn and Shin [1]		✓			
AuthUML [2]		✓			

represent the main areas of security engineering in the systems security engineering capability maturity model (SSE-CMM) on which this methodology is based. A security design pattern is used to specify each of those processes.

From the state of the art, three main UML artifacts can be used for security specification: (1) stereotypes and tagged values, (2) OCL, and (3) behavior diagrams. In addition, two other approaches can be used: (1) extending the UML metalanguage or (2) creating a new metalanguage. Table 1 summarizes the use of these approaches to specify security requirements by the contributions presented above.

In the following, we present each of those approaches and explain how it can be used for security specification. The activity diagram of Figure 1 will be used throughout the following subsections to show how security requirements can be specified for UML design. The diagram specifies the behavior related to the admission of patients in a medical institution. This example is a simplified version of the business process used in [24]. The activity diagram consists of three main partitions: (1) Patient who starts the activity by filling out an admission request, (2) Administration area where insurance and cost information are collected, and (3) Medical area which is responsible for admission tests, exams, medical evaluations and sending the medical results to the patient.

Security Specification Using UML Artifacts

In this section we show how stereotypes and tagged values, the OCL, and behavior diagrams can be used for security specification and design.

Stereotypes and Tagged Values

Description: *Stereotypes* are provided as a mechanism for extending the UML meta-language. Therefore, a stereotype is considered as a user-defined meta-element. Its structure matches the structure of an existing UML meta-element which is referred to as “base class”. In that sense, a stereotype represents a subclass (subtype) of the base class. It has the same form but with a different intent. A stereotype can

have tagged values used to define the additional information needed to specify the new stereotype intent. Besides, constraints can be defined on both the base class attributes as well as the tagged values. Code generators and other tools, such as those used for verification and validation, reserve special treatment to stereotypes.

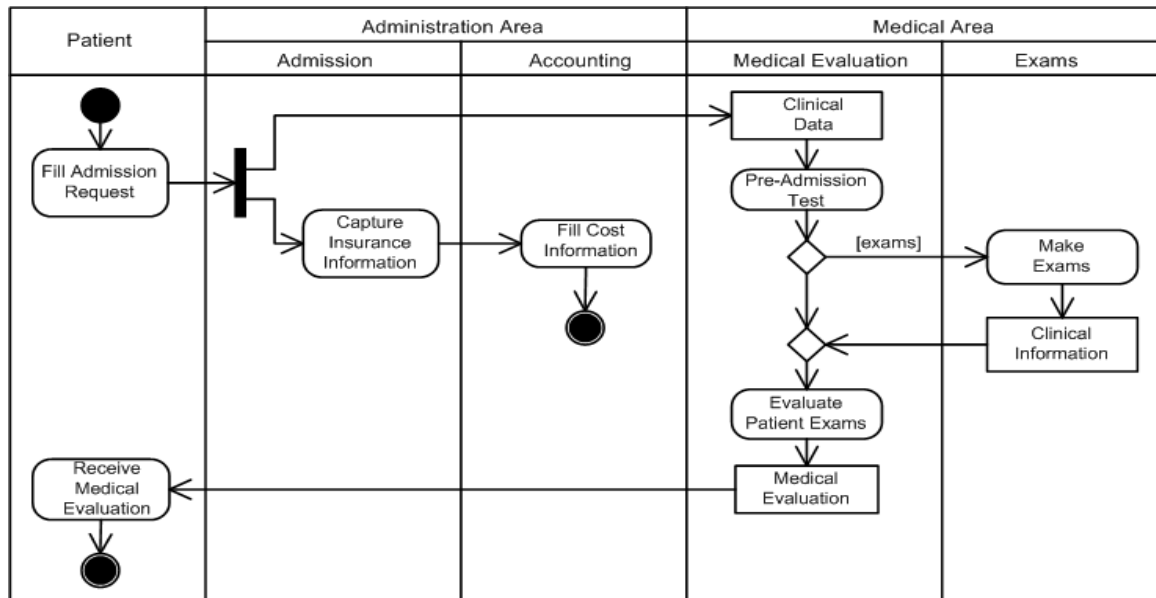
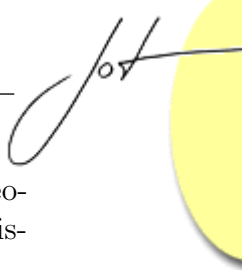


Figure 1: An Activity Diagram: Admission of Patients in a Medical Institution.

Use for Security Specification: Security requirements are specified by attaching stereotypes along with their associated tagged values to selected elements of the design (e.g., subsystems, classes, etc.). Thus a “security” profile should be created by some security expert for the specification of these stereotypes. The compiler used to parse UML diagram is then modified such that it can read and interpret the stereotypes annotating the design. This interpretation consists in generating a formal representation of the security requirement corresponding to the security annotation. This security requirement is generated on the basis of the intent of the security expert while taking into consideration the specificities of each design. In addition, a formal semantics is associated with the design. Then, the formal security requirement together with the formal semantics are provided as inputs to a verification tool (usually a model checker or a theorem prover). The result of verifying the security requirement on the design is translated into some representation that any non-security expert developer can understand. Some stereotypes are parameterized over the adversary type. These stereotypes are used to specify security properties that need to be verified against a specification of an attacker (adversary). Fair exchange, secrecy, and authenticity are examples of these properties. The adversary type specifies the adversary’s computation capabilities and initial knowledge.

Figure 2 shows how stereotypes can be used to specify security requirements on the UML design of Figure 1. The used stereotypes are Privacy, Auditing,



Access Control, Critical, Integrity, and NonRepud. For example, the stereotype `Privacy` is attached to the `Patient` partition to specify that unauthorized disclosure of sensitive information about the patient is not permitted.

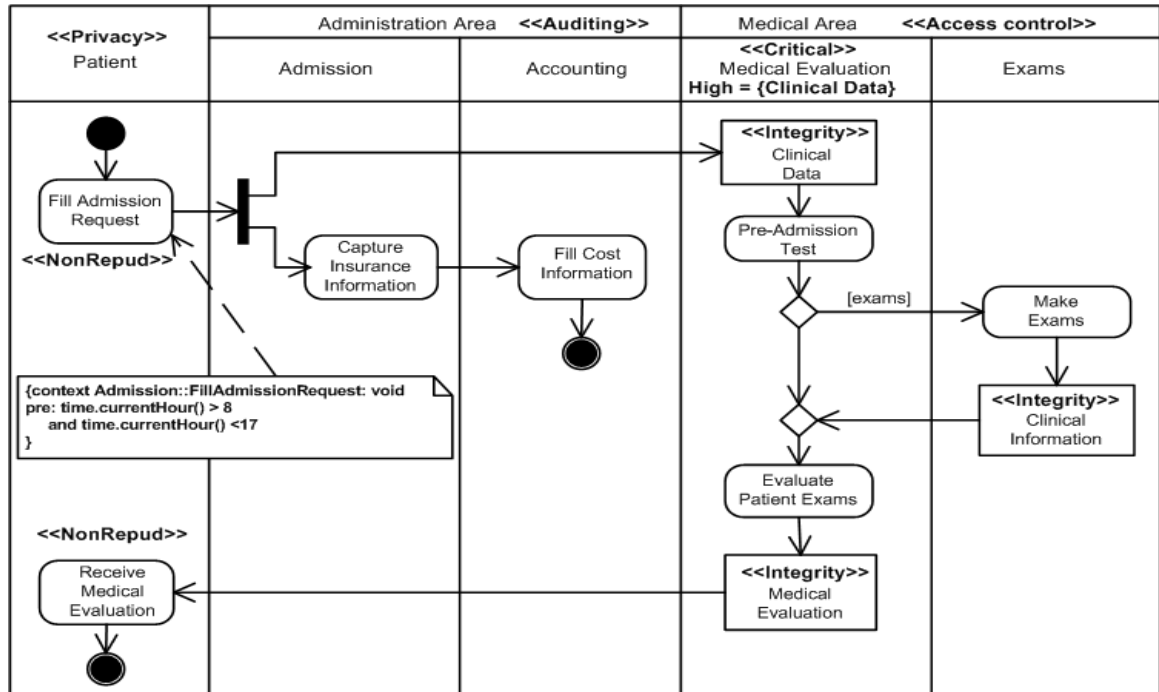


Figure 2: An Example of Specifying Security Using Stereotypes.

Object Constraint Language (OCL)

Description: The OCL is a formal language used to express constraints over UML diagrams. These constraints mainly specify those conditions that must be satisfied by the system being modeled. The OCL is mainly used to specify application-specific requirements for UML models. In addition it is used to specify invariants of the UML meta-language. More precisely, the main purposes for which OCL can be used are the followings: (1) To specify invariants on classes and types in the meta-language, (2) to specify type invariant for Stereotypes, (3) to describe pre and post conditions on operations and methods, and (4) to describe guards [11].

Use for Security Specification: Since OCL is a language for constraints specification, it is natural to be used for security specification. According to the main usability purposes listed above, OCL has been used for security specification following three main directions. First, for the security profiles extending UML for security specification, OCL is used to define constraints on elements described by stereotypes and tagged values. Second, for those stereotypes used for the specification of access control properties, OCL can be used by the designer to define access control constraints (pre conditions and authorization guards). Third, some OCL extensions

[29] allow the specification of temporal logic formulas and thus are used to specify security requirements in temporal logics, e.g., LTL, CTL, etc. Figure 2 shows how OCL can be used to specify a constraint on the action “Fill admission request”. This constraint restricts the execution of this action to the working hours. This will protect the system from malicious use during nights. The condition start by specifying its context, i.e., the method on which it is applied, which is the method `FillAdmissionRequest` of the class `Admission`. Then the constraint specifies the pre condition to be satisfied before executing the controlled method.

Behavior Diagrams

Description: Behavior diagrams are UML diagrams used to depict the behavior features of the system under design. These include activity, state machine, and use case diagrams as well as four interaction diagrams. The later are those diagrams used to specify interactions between objects inside the system. Interaction diagrams include communication, interaction overview, sequence, and timing diagrams.

Use for Security Specification: Behavior diagrams can be used for security specification in two ways. The first one is to specify the behavior that ‘MUST’ be observed by the system and the second one is to specify the behavior that ‘MUST NOT’ be observed by the system. The later has been investigated by some recent contributions [30] where the used diagrams are called “Abuse cases diagrams”. Figure 3 shows an example of an activity diagram specifying the behavior that must be followed by the system after filling the cost information until sending the medical evaluation to the patient. This behavior is required for enforcing faire exchange between patients and the medical institution. Enforcing this behavior inside the original design of Figure 1 results to the new design presented in Figure 4. This represents one possible scenario of using behavior diagrams to enforce security requirements. A non-security expert designer will use this “safe design” and integrate it inside its original design. Another possible scenario is when the behavior diagram, specifying a security requirement, is used to verify, through model checking or theorem proving, whether the design satisfies or not the security requirement. In this case, the diagram is translated into a (1) transition system (finite state machine or automata, etc.) or (2) a logic formula, both expressed in the input language of the target verification tool. Indeed, many contributions establishing the correspondence between transition systems and temporal logics can be found in language theory [3]. A third possible scenario is the use of behavior diagram to specify security aspects. Indeed, aspects [15] are usually defined by specifying a behavior that is inserted before or after some execution point. Thus this behavior can easily specified by a behavioral diagram. However, the weaving of aspects and the original design can be performed on the level of design by weaving UML diagrams or postponed to the implementation phase. In the later case, the weaving is performed on selected files of the source code and the actual aspects expressed in existing aspect languages, e.g., AspectJ, and resulting from the refinement of their initial behavior diagrams.

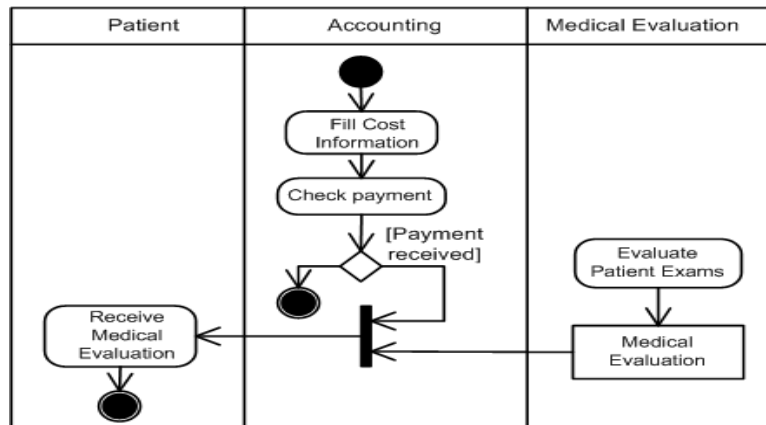


Figure 3: Fair Exchange Requirement Inside Medical Applications.

Security Specification by Extending the UML Meta-language

This section shows how the UML metalanguage can be extended to specify security. **Description:** In this approach, the UML meta-language is directly extended by a meta-language specification language as MOF (the Meta-Object Facility) [10]. The MOF defines a simple meta-metamodel, and the associated semantics, allowing the description of metamodels in various domains including the domain of object design and analysis. Extending the UML meta-language (meta-model) is usually needed when extension mechanisms provided by UML (mainly stereotypes) are not appropriate for the target extension or when the resulting complexity is not tolerated.

Use for Security Specification: The two reasons stated above are the same motivating the extension of UML meta-language for security specification. Although, stereotypes allow the specification of a wide range of security requirements, they are not appropriate for specifying structured security policies: Those that are usually specified using well structured specification languages. Access control properties and security aspects are the main requirements for which it is better to have dedicated meta-elements than using standard UML meta-elements annotated by stereotypes and tagged values.

Security Specification by Creating New Meta-languages

This section shows how new metalanguages can be proposed for specifying security.

Description: In this approach, a new meta-language is defined using a meta-language specification language as MOF. The motivations of crating a new meta-language are the same as those of extending the UML meta-language. The vocabulary used by the meta-elements defined by the new meta-language have domain-specific intuition and are much more precise than the one used for UML meta-elements. Thus, the interfaces needed for manipulating the new meta-elements are

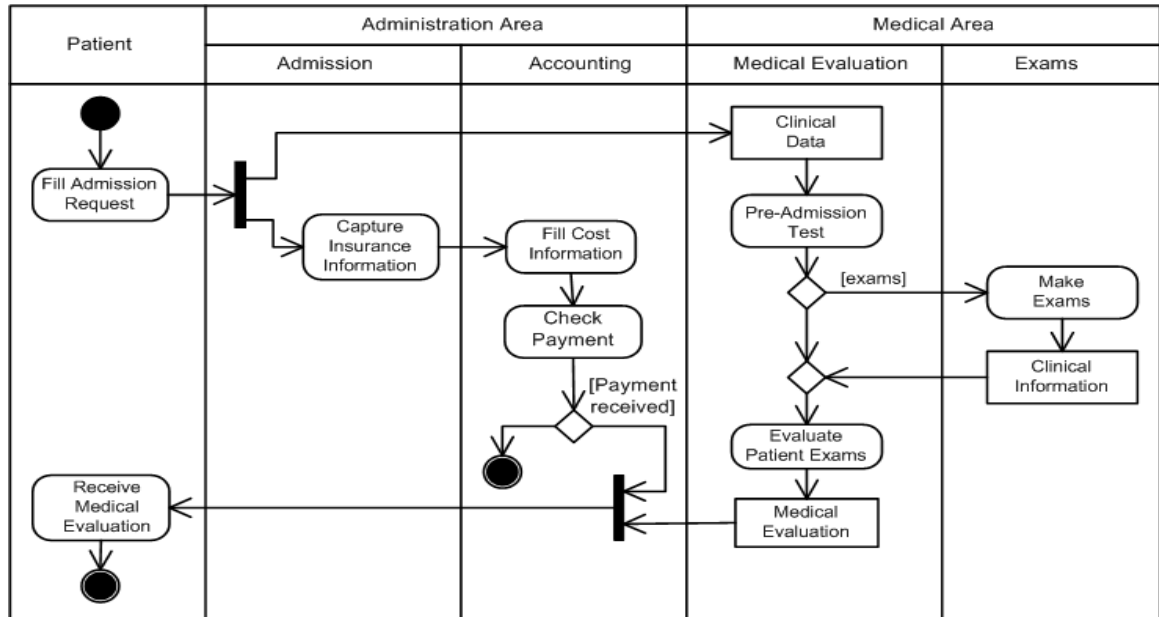


Figure 4: Enforcing the Security Requirement of Figure 3 in the Activity Diagram of Figure 1.

too simpler compared to the those required for UML design.

Use for Security Specification: The motivations of creating new meta-languages for security specification are exactly the same of extending the UML meta-language for security specification. Indeed, the approach is used for the same objectives and allows the specification of almost the same security requirements.

3 USABILITY DISCUSSION

This section discusses the usability of each security specification approach on the light of our survey of the state of the art. First we define a set of usability criteria that will be used later to discuss the different security specification presented in the previous section. For the first approach, we discussed separately the usability of each of the three UML artifacts used for security specification. The results of the usability discussion of all the approaches are summarized in Table 2.

Usability Criteria

Inspired from the state of the art of software usability and software security requirements specification, we defined the following usability criteria:

- *Expressiveness*: Refers to the ability of specifying security requirements. This



is a discriminatory criterion since it leads to the rejection of any approach that fails to specify the desired security requirements. Regarding this criterion, any specification approach will be given two ranks. The first rank is related to the covered security requirements and can take one of the following values:

- *Static_Reqs*: if it allows the specification of the majority of statically enforceable security requirements.
- *Dynamic_Reqs*: if it allows the specification of the majority of dynamically enforceable security requirements.
- *All_Reqs*: if it allows the specification of almost all the security requirements.

The second rank is related to the logic classification of the security requirements that can be specified. It can take one of the following values:

- *LTL_Logic* if security requirements belonging to LTL logic can be specified by the evaluated approach.
 - *CTL_Logic* if security requirements belonging to CTL logic can be specified by the evaluated approach.
 - *All_logics* if the specifiable security requirements can belong to any logic class.
- *Tool Support*: Refers to the availability of tools for specification and verification of security requirements, which is of paramount importance. Tools are mainly used to (1) artifacts the specification and (2) compile and store the specification in a useful intermediate representation (for verification and/or code generation). Tool support will be ranked by using one of the following three values: (1) *Standard* when tools are provided by any standard UML modeling framework, (2) *HighlyPortable* when they are not supported by all standard UML frameworks but can be easily ported (e.g., plugged in), or (3) *WeaklyPortable* when the tools are almost unportable.
 - *Verifiability*: Refers to the efforts needed to verify the design against the security requirements. These cover (1) associating a semantics to the UML design, (2) formally specifying security requirements, (3) actually verifying the design against the security requirements, and (4) interpreting and presenting the verification results. Verifiability will be ranked using one of the following three values: (1) *Comp_Verif* for complex verifiability, (2) *Good_Verif* for verifiability with acceptable efforts, and (3) *Ease_Verif* for ease verifiability.
 - *Complexity*: Refers to the amount of security-relevant information added to a UML design and its impact on its readability. Complexity will be evaluated using one of the following three values: (1) *High_Comp* when the added security information seriously deteriorates the readability of the design, (2) *Acceptable_Comp* when the it is tolerated, or (3) *Low_Comp* when it is negligible compared to the original design complexity.

Security Specification Using UML Artifacts

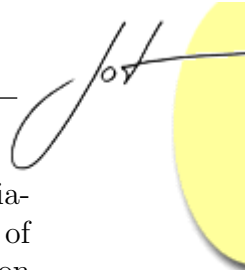
Stereotypes and Tagged Values:

In the following we discuss the usability of stereotypes and tagged values for security specification.

- *Expressiveness*: UML artifacts provided by standard UML mainly stereotypes and tagged values are the most used by the majority of the contributions. Among these contributions, we can cite: UMLSec [14] by Jürjens which provides a UML profile and an open-source tool for specifying security requirements such as secrecy, integrity, authenticity, fair exchange, role-based access control, secure communication links, and secure information flow. Stereotypes are used by Pavlich-Mariscal *et al.* [20] and Basin *et al.* [17] for specifying access control policies and by Montangero *et al.* [18] for modeling authentication protocols. These contributions show that various security requirements have been specified using stereotypes and tagged values.
- *Tool Support*: Has an excellent tool support since any standard UML modeling framework supports profile specification.
- *Verifiability*: A lot of work is done in background to generate a formal semantics for the UML design, formally specify the security requirement, verify the property against the design, and show the verification result to the end user (UML designer). The later usually consists in displaying counter examples and providing advices to improve the design and fix the vulnerabilities.
- *Complexity*: The complexity of the information related to stereotypes and tagged values added for security specification, depends on the number of stereotypes and tagged values attached to each UML element. For example, if different security stereotypes are associated with the same UML element then it will be complex for the user to select all these stereotypes and edit the associated tagged values. In this case, the security profile designer has the responsibility of compacting as possible the architecture of his profile design.

OCL

The OCL is also used by many of the surveyed contributions to express formal constraints in the specification of security properties. This is due to the fact that OCL is part of the UML standard, and by its formal nature, it allows precise specification of security constraints. The approach of Painchaud *et al.* (SOCLe project) [19] is based on temporal logic extension of OCL for security specification. OCL has been also used by [17] to specify additional authorization constraints related to the state of the system. As we mentioned above, it is natural to use OCL for security specification. However, it is important here to distinguish between using OCL as



a support for some security specification artifact as stereotypes and behavior diagrams, and using it as security specification language. In the former case, the use of OCL improves the usability of any specification artifact by allowing the definition of constraints over the UML design entities. Accordingly, we focused our usability evaluation on the later case. In the following we discuss the usability of OCL for security specification.

- *Expressiveness*: As a security specification language, the standard OCL [11] is limited to specifying pre and post conditions and invariants that should be satisfied by the application behavior. However, some OCL extensions allow the specification of temporal logic properties.
- *Tool Support*: Standard OCL benefits from the support of different tools provided by standard UML modeling frameworks. However, the usability of OCL extensions is limited by the availability of tools supporting the specification and the compilation of security requirements.
- *Verifiability*: Once compiled and analyzed by the tool, security requirements specified using OCL extensions are systematically provided as input formulas for verification tools (model checkers and/or theorem provers). However, as for stereotypes, a lot of work is done in background to generate a formal semantics for the UML design, verify the properties against the design, and show the verification result to the end user (UML designer).
- *Complexity*: The complexity introduced by this approach depends on the number of OCL expressions added to specify security properties and whether they are crosscutting the application functionalities design or separated from them.

Behavior Diagrams

We notice the lack of using behavioral diagrams for security specification among the surveyed approaches. In fact, only the approach of Zisman *et al.* [30] that proposes the modeling of abuse cases to represent possible attack scenarios and potential threats to the system security. In the following we discuss the usability of behavior diagrams for security specification. We distinguish in our discussion between the use of behavior diagrams to specify security requirements for the sake of verification and their use to specify security aspects for the sake of security enforcement or hardening.

- *Expressiveness*: Is limited to specify those security requirements that are naturally expressible via transition systems. These include mainly attack scenarios and dynamically enforceable security requirements. As for security aspects specification, behavior diagrams are very useful for specifying advices behavior. However, stereotypes should be defined to allow the specification of patterns needed for the definition of pointcuts.

- *Tool Support*: Behavior diagrams benefit from a wide tool support. However, tool support for this approach depends also on the tool support of stereotypes.
- *Verifiability*: When used for security requirements specification, behavior diagrams are translated to transition systems or logical formulas in order to be verified on the system design. While the former translation is almost systematic, the later is limited to those diagrams satisfying some structural constraints (e.g., determinism) and constrained by the availability of translation algorithms in language and logic theory. As for stereotypes and OCL, a lot of work is done in background to generate a formal semantics for the UML design, verify the properties against the design, and show the verification result to the end user (UML designer). When used for security aspects specification, as for the first approach, a lot of work is done in background to (1) identify diagram entities (e.g., methods/actions) matching the specified patterns and (2) weaving diagrams specifying advices and those specifying the system behavior.
- *Complexity*: Relatively acceptable since the behavior diagrams specifying security requirements are separated from those specifying the system behavior and are easily distinguishable from them. The complexity of security aspects specification is comparable to that of security requirements specification.

Extending the UML Metalanguage

Only few contributions [20] have investigated the extension of the UML metalanguage for security specification. This is due to the fact that this kind of modification requires a high expertise and knowledge of the UML meta-language and its objectives. Indeed, the extension may require the modification of the whole meta-language which is too complex. In the following we discuss the usability of extending the UML metalanguage for security specification.

- *Expressiveness*: Comparable to that of stereotypes.
- *Tool Support*: The extension is heavyweight so that “may require one to extend the CASE-tool itself, in particular the storage components, i.e., the repository, and the visualization components” [17]. This impacts negatively the portability of any extension since any UML modeling framework is heavily modified to allow the use of the new meta-elements and their interpretation.
- *Verifiability*: A lot of work is done in background to generate a formal semantics for the UML design, verify the properties against the design, and show the verification result to the end user. However, if the extension targets some low-level policy specification language or AOP language, then the effort spent in background is limited to parsing the specification and translating it to the target language.
- *Complexity*: The complexity is comparable to that of using behavior diagrams.

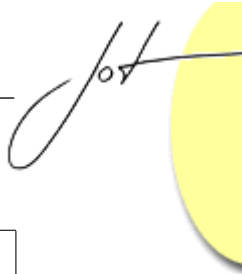


Table 2: Usability Evaluation of Security Specification Approaches.

	Stereotypes and tagged values	OCL	Behavior Diagrams	Extending UML Meta-language	Creating New Meta-languages
Expressiveness	<i>All_Reqs, All_Logics</i>	<i>All_Reqs, All_Logics</i>	<i>Static_Reqs, LTL_Logic</i>	<i>Static_Reqs, All_Logics</i>	<i>All_Reqs, All_Logics</i>
Tool Support	<i>Standard</i>	<i>HighlyPortable</i>	<i>Standard</i>	<i>HighlyPortable</i>	<i>WeaklyPortable</i>
Verifiability	<i>Comp_Verif</i>	<i>Good_Verif</i>	<i>Good_Verif</i>	<i>Good_Verif</i>	
Complexity	<i>Low_Comp</i> to <i>High_Comp</i>	<i>Acceptable_Comp</i>	<i>Low_Comp</i>	<i>Acceptable_Comp</i>	<i>Acceptable_Comp</i>

Creating a New Metalanguage

As for the previous approach, only few contributions [17] have investigated the creation of new meta-languages for security specification. In the following we discuss the usability of creating a new metalanguage for security specification.

- *Expressiveness*: Comparable to the expressiveness of extending the UML metalanguage.
- *Tool Support*: Better than that of extending the UML metalanguage and comparable to that of stereotypes. In addition, the compiler needed to parse the specification can be easily plugged in to the UML modeling framework.
- *Verifiability*: Better than that of the verifiability of extending the UML metalanguage. Indeed, the security specification is exclusively based on the new meta-elements and thus is easier to parse and translate.
- *Complexity*: Comparable to the complexity of extending the UML metalanguage.

4 CONCLUSION

This paper investigates the main approaches adopted for specifying and enforcing security requirements at UML design and surveys the related state of the art. The main contribution of this paper is a discussion of these approaches from usability viewpoint. We distinguish between those approaches that are based on the artifacts provided by the standard UML specification and those that require explicit extension of the UML meta-language. This mainly allow one to understand when it is better to use UML artifacts and when it is useful to extend the its meta-language. We defined a set of criteria that we used in our usability discussion. In the following we provides the conclusions of our discussion:

- Stereotypes are the most usable for security specification since they are the extension mechanism provided by UML. They allow the specification of almost

all security requirements that are usually specified and enforced on software. In addition, they are easy to learn and use and benefit from a high portability. However, it is difficult to associate a formal semantics to stereotypes which can affect the verifiability of the specified security requirements.

- Using OCL for security specification is not a good approach since the security requirements that can be expressed using OCL are limited to pre and post conditions and invariants. Although OCL can be extended to allow the specification of security requirements in temporal logics, it remains more usable as a complementary capability for stereotypes. We strongly believe that temporal logic should be used on background and not exposed to UML designers.
- UML behavior diagrams seem to be extremely usable for security requirements specification and aspect design. However, few efforts have been spent in the literature to investigate their use.
- Extending the UML meta-language is a too constraining approach, though it has its motivation. Creating a new meta-language should be then an appropriate alternative.
- The efforts needed for verifying the design against a security requirement is often very important regardless of the adopted approach.

REFERENCES

- [1] G-J. Ahn and M. E. Shin. Role-based authorization constraints specification using object constraint language. In *WETICE '01: Proceedings of the 10th IEEE International Workshops on Enabling Technologies*, pages 157–162, Washington, DC, USA, 2001. IEEE Computer Society.
- [2] K. Alghathbar and D. Wijeskeru. Consistent and complete access control policies in use cases. In *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, pages 373–387, October 2003.
- [3] A. Bogdanov, S. J. Garland, and N. A. Lynch. Mechanical translation of i/o automaton specifications into first-order logic. In *In proceedings of the 22nd IFIP WG 6.1 International Conference Houston*, pages 364–368, 2002.
- [4] G. Brose, M. Koch, and K. P. Lohr. Integrating access control design into the software development process. In *Proceedings of the sixth biennial world conference on the Integrated Design and Process Technology, IDPT*, Pasadena, CA, June 2002.
- [5] M-T. Chan and L-F. Kwok. Integrating security design into the software development process for e-commerce systems. *Information Management & Computer Security*, 9(3):112–122, 2001.



- [6] T. Doan, L. D. Michel, and S. A. Demurjian. A formal framework for secure design and constraint checking in UML. In *Proceedings of the International Symposium on Secure Software Engineering, ISSSE'06*, Washington, DC, Mars 2006.
- [7] P. Epstein and R. S. Sandhu. Towards a UML based approach to role engineering. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 135–143. ACM Press, 1999.
- [8] E. B. Fernández. A methodology for secure software design. In *Software Engineering Research and Practice*, pages 130–136, 2004.
- [9] M. Gogolla and B. Henderson-Sellers. Analysis of uml stereotypes within the uml metamodel. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 84–99, London, UK, 2002.
- [10] Object Management Group. Meta object facility (mof) specification, version 2.0, 2006.
- [11] Object Management Group. Uml 2.0 ocl specification, version 2.0, 2006.
- [12] Object Management Group. Unified modeling language: Superstructure, version 2.1.2, 2007.
- [13] A. Hamie, F. Civello, J. Howse, S. Kent, and R. Mitchell. Reflections on the Object Constraint Language. In *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse, France*, pages 137–145, 1998.
- [14] J. Jürjens. *Secure Systems Development with UML*. Springer Verlag, 2004.
- [15] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.
- [16] Y. Ledru, R. Laleau, M. Lemoine, S. Vignes, D. Bert, V. Donzeau-Gouge, C. Dubois, and F. Peureux. An attempt to combine UML and formal methods to model airport security. In *Forum of the 18th International Conference on Advanced Information Systems Engineering*, pages 47–50, Luxembourg, 2006.
- [17] T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the International Conference on the Unified Modeling Language, UML'2002*, pages 426–441, 2002.
- [18] C. Montangero, M. Buchholtz, L. Perrone, and S. Semprini. For-lysa: UML for authentication analysis. In *Global Computing: IST/FET International Workshop, GC'2004*, pages 93–106, 2005.

- [19] F. Painchaud, D. Azambre, M. Bergeron, J. Mullins, and R. M. Oarga. Socle: Integrated design of software applications and security. In *Proceedings of the 10th International Command and Control Research and Technology Symposium, ICCRTS'2005*, McLean, VA, USA, 2005.
- [20] J. Pavlich-Mariscal, L. Michel, and S. Demurjian. Enhancing UML to model custom security aspects. In *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling*, 2007.
- [21] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.
- [22] G. Popp, J. Jürjens, G. Wimmel, and R. Breu. Security-critical system development with extended use cases. In *Proceedings of the 10th Asia-Pacific Software Engineering Conference, APSEC*, pages 478–487, 2003.
- [23] I. Ray, N. Li, D. K. Kim, and R. France. Using parameterized UML to specify and compose access control models. In *Proceedings of the 6th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems, IICIS'03*, Lausanne, Switzerland, November 2003.
- [24] A. Rodriguez, E. Fernandez-Medina, and M. Piattini. Security requirement with a uml 2.0 profile. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 670–677, Washington, DC, USA, 2006. IEEE Computer Society.
- [25] A. Schleicher and B. Westfechtel. Beyond stereotyping: Metamodeling approaches for the uml. In *HICSS*, 2001.
- [26] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and Systems Security*, 3(1):30–50, 2000.
- [27] E. Song, R. Reddy, R. France, I. Ray, G. Georg, and R. Alexander. Verifiable composition of access control and application features. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, SACMAT'05*, pages 120–129. ACM, 2005.
- [28] J. L. Vivas, J. A. Montenegro, and J. Lopez. Towards a business process-driven framework for security engineering with the UML. In *Proceedings of the 6th Information Security Conference, ISC'03*, pages 381–395, Bristol, U.K., 2003.
- [29] P. Ziemann and M. Gogolla. An extension of ocl with temporal logic. pages 53–62, September 2002.
- [30] A. Zisman. A static verification framework for secure peer-to-peer applications. In *Second International Conference on Internet and Web Applications and Services, ICIW'07*, page 8, 2007.