Verification and test of interoperability security policies

Mazen El Maarabani

TÉLÉCOM SUDPARIS

En co-accréditation avec l'université d'Evry

École Doctorale - S&I

Verification and Test of Interoperability Security Policies

Thèse de Doctorat

Mention : **Informatique**

Présentée par **Mazen El Maarabani**

Directeur de thèse : **Ana Cavalli**

Soutenue le 29 Mai 2012

Composition du Jury

Rapporteurs :
- Nora Cuppens-Boulahia, Enseignant-chercheur, Télécom Bretagne
- Mercedes G. Merayo, Enseignant-chercheur, Universidad Complutense de Madrid

Examinateurs :
- Frédéric Cuppens, Professeur, Télécom Bretagne
- Fatiha Zaïdi, Enseignant-chercheur, Université Paris-Sud XI
- Ana Cavalli, Professeur, Télécom SudParis

N d'ordre :2012TELE0016

"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair."

Douglas Adams

Dedicated to my tender mother, awesome father,
lovely sisters and genuine brother.

Abstract

NOWADAYS, there is an increasing need for interaction in business community. In such context, organizations collaborate with each other in order to achieve a common goal. In such environment, each organization has to design and implement an interoperability security policy. This policy has two objectives: *(i)* it specifies the information or the resources to be shared during the collaboration and *(ii)* it define the privileges of the organizations' users. To guarantee a certain level of security, it is mandatory to check whether the organizations' information systems behave as required by the interoperability security policy. In this thesis we propose a method to test the behavior of a system with respect to its interoperability security policies. Our methodology is based on two approaches: active testing approach and passive testing approach. We found that these two approaches are complementary when checking contextual interoperability security policies. Let us mention that a security policy is said to be contextual if the activation of each security rule is constrained with conditions.

The active testing consists in generating a set of test cases from a formal model. Thus, we first propose a method to integrate the interoperability security policies in a formal model. This model specifies the functional behavior of an organization. The functional model is represented using the Extended Finite Automata formalism, whereas the interoperability security policies are specified using OrBAC model and its extension O2O. In addition, we propose a model checking based method to check whether the behavior of a model respects some interoperability security policies. To generate the test cases, we used a dedicated tool developed in our department. The tool allows generating abstract test cases expressed in the TTCN notation to facilitate its portability.

In passive testing approach, we specify the interoperability policy, that the system under test has to respect, with Linear Temporal logics. We analyze then the collected traces of the system execution in order to deduce a verdict on their conformity with respect to the interoperability policy.

Finally, we show the applicability of our methods though a hospital network case study. This application allows to demonstrate the effectiveness and reliability of the proposed approaches.

Keywords: interoperability security policies, OrBAC, O2O, security policies verification, security policy testing, active testing, passive testing.

Resumé

De nos jours, divers systèmes ou organisations peuvent collaborer et échanger des informations ou des services. Ainsi grâce à cette collaboration, ces derniers vont pouvoir travailler ensemble et mener des échanges afin d'atteindre un but commun. Ceci ne peut pas être réalisé sans des problèmes de sécurité. Pour collaborer chaque participant doit définir une politique d'interopérabilité. Cette politique sera en charge de : (i) définir les informations et les ressources partageables et (ii) définir les privilèges d'accès des utilisateurs qui participent à un projet commun qui nécessite une collaboration. Pour garantir un certain niveau de sécurité, il est indispensable de vérifier si le comportement des systèmes des différents participants respecte bien leurs politiques de sécurité. Pour atteindre cet objectif, nous proposons une méthode pour tester les politiques d'interopérabilité en se basant sur deux approches différentes de test : l'approche active et l'approche passive. Le principe de test actif consiste à générer automatiquement une suite de scenarios de test qui peuvent être appliqués sur un système sous test pour étudier sa conformité par rapport à ses besoins en matière de sécurité. Quant au test passif, il consiste à observer passivement le système sous test sans interrompre le flux normal de ses opérations. Dans notre étude nous avons remarqué que les techniques de test actif et passif sont complémentaires pour tester les politiques d'interopérabilité contextuelles. Une politique est dite contextuelle si l'activation de chacune de ses règles est conditionnée par des contraintes qui peuvent être liées à l'environnement de la collaboration ou à chaque participant.

Afin de pouvoir générer automatiquement les scenarios de test, il est indispensable de modéliser les politiques d'interopérabilité et le comportement fonctionnel des participants. Dans cette thèse, nous proposons une méthode pour intégrer les politiques d'interopérabilité dans les modèles fonctionnels des participants afin d'obtenir un modèle sécurisé des participants. Le comportement fonctionnel des participants est modélisé par un modèle formel basé sur des automates à états finis. Tandis que les besoins de sécurité sont spécifiés en utilisant le modèle formel OrBAC et son extension O2O. De plus, nous proposons une méthode fondée sur la technique de model checking pour vérifier si le comportement des modèles utilisés dans notre processus de test respecte bien les politiques de sécurité. La génération de cas de test est ensuite effectuée en utilisant un outil développé dans notre laboratoire. Cet outil permet d'obtenir des cas de test abstraits décrits dans des notations standards (TTCN) facilitant ainsi leur portabilité. Dans l'approche de test passif, nous spécifions la politique d'interopérabilité que le système doit respecter en utilisant un langage temporel de premier ordre. Nous analysons ensuite les traces d'exécutions des participants afin d'élaborer un verdict sur leur conformité par rapport à la politique d'interopérabilité.

Finalement, nous avons appliqué nos méthodes sur un cas d'usage d'un réseau hospitalier. Cette application permet de démontrer l'efficacité et la fiabilité des approches proposées.

Mots-clés : politique de sécurité d'interopérabilité, modélisation des politiques de sécurité, vérification des politiques de sécurité, OrBAC, test des politiques de sécurité, test actif, test passif.

Acknowledgements

I am deeply grateful to my supervisor, Prof. Ana Cavalli for her detailed and constructive comments, and for her important support throughout this work. I will never forget all the extensive discussions we had on diverse topics. It is always interesting to learn her way of thinking and his point of view.

I would like to express my deep and sincere gratitude to Dr. Iksoon Hwang. His wide knowledge and his logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a crucial and solid basis for the present thesis.

I would also like to thank Dr. Anis Laouiti, Dr. Amel Mammar and Dr. Stephan Maag lecturers at the Telecom SudParis institute, for their precise explanations and criticisms. Their constructive remarks helped me to build my personal point of view.

During this work, I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to Dr. Faycal Bessayah, Dr. bakr Sarakbi, Mme. Afef Sayadi, Mr. Anderson Morais, Dr. Felipe Lalanne, and Mr. Khalifa Toumi who helped me generously in this work.

I would like to thank deeply Prof. Frédéric Cuppens, Dr. Mercedes G. Merayo, Dr. Nora Cuppens-Boulahia, Dr. Fatiha Zaïdi, and Mr. Philippe Rose, who examined my work and accepted to participate to the Jury: it was a great privilege and honor for me.

I would also like to convey thanks to Télécom SudParis for providing all the necessary means and laboratory facilities in order to complete successfully this work.

Last but not least, I owe my loving thanks to my family and my closest friends. Without their encouragement and understanding, it would have been impossible for me to finish this work.

Finally, all mistakes in this thesis are mine.

Contents

List of Figures

List of Tables

"Keep me away from the wisdom which does not cry, the philosophy which does not laugh and the greatness which does not bow before children. "

Gubran Khalil Gubran

Chapter 1

Introduction

Contents

1.1 Motivation

Nowadays, there is an increasing need for interaction in business community. The insufficiency of financial needed urges organizations to share resources. For instance we have cloud computing and hospital networks. In addition, organization may find benefits in sharing information in order to achieve common goals. In such context, organizations collaborate with each other in order to create a Multi-Organization Environment. This environment is a set of distributed systems that interact with each others where this architecture arises some security problems.

Before starting to work together, the organizations have to describe their isolated roles. Each has its own mission, vision, information and resources, as well as its own distinct boundaries. Even when they are in the same sector or field, which might represent a basis for common interest or action, many organizations remain solely focused on shaping their own position in the sector or field. Inside these boundaries, the security of information and resources regarding the local users can be specified by using local security policies. Sometimes organizations will find that there is value in coordinating efforts. Indeed, in a collaborative environment the boundaries fade away. In such context, each organization has to be able to distinguish between critical information or resources and those that could be shared. For instance, when government entities consider legislation or regulation on fuel efficiency or driver safety regulations, automakers may decide that they will present a common message on the cost of regulations to the consumer, industry and workforce. In the process of arriving at the common message they will probably not share information about how their engineers are designing fuel efficiency or safety improvements that will profit the company in the future. Accordingly, each organization, in addition to its local

policies, specifies and implements an *interoperability security policy*. There are two goals in this policy on the one hand to specify the information and resources that can be shared and on the other hand to define the access rights of external users to these information and resources.

Once the interoperability security policy is implemented in a system, one has to decide if this system is behaving as required by its security policies. This interesting issue has been driving researches to propose several solutions in order to reach this objective. Several techniques have been proposed in this area and they can be divided into two categories: active testing techniques and passive testing techniques. In active testing techniques the process passes through three main steps. In the first step, the functional aspect of the system and its interoperability security policies are described using formal models. In the second step we check whether the behavior of this model conforms the interoperability security policies. Without this step, we cannot guarantee whether this model respects the requirements to be tested. Therefore it will not be possible to rely on this model for performing formal test. In the third step, the model is used to check the conformance of the system with respect to its security policy specification. The latter can be verified by (1) injecting this policy in the system or (2) specifying formally the target system and providing proofs that this system implements the security policy or (3) considering several strategies of formal tests. In contrast, passive testing consists of observing the input and output events of a system under test at run-time. These techniques should not disturb the execution of the system, that is why it is called passive testing. The record of the event observation is called an event trace. This event trace will be analyzed according to some security requirements in order to determine the conformance relation between the implementation and these security requirements. We should keep in mind that if an event trace conforms to these requirements it does not means that the whole implementation conforms to these requirements.

Recently, the diversity and different requirements of the information system have led to the development of a variety of increasingly expressive policy specification languages. For instance, these languages are not restricted to permissions anymore. A security policy may include different types of security rules such as permission, prohibition and obligation. A prohibition is a negative permission implying that one must not perform some action. An obligation is associated with an action that someone must perform and is usually triggered when some conditions are satisfied. In addition, the increasing complexity in the requirements of secured interactions between systems urges security models to handle more flexible and dynamic security policies [CCB08]: security rules in these policies are no longer static but dynamic, depending on the context. The advantage of such dynamicity is to provide a self-adaptive access control security policy that is aware of changes in the interoperability environment.

In this Thesis, we take into consideration these security policy requirements in our security testing process. Furthermore, we consider the system under test as a distributed system that is composed of at least two systems that are interacting based on interoperability security policies. Our aim is to provide a framework to formally test systems with respect to its interoperability security policy.

1.2 Contributions

The main core of our contribution is combining active based testing in a passive based techniques methodology, to provide a framework for testing interoperability security policies. Indeed, we need first to be able to model interoperability security policies. The existing modeling techniques have to be properly extended to adapt the interoperability security policies exigencies. In the state of the art we can find two approaches for modeling security policies. In the first one, security policies are modeled using formal models. In this case, these models are used to verify security properties such as consistency, completeness and termination [HK10]. Other work, for instance [BM04], model the functional aspect of a system and its security policies using two different models. These two models interact to deliver a behavior that respects these security policies. The other approach [MOC+07] consists at integrating the security policy in the functional aspect of the system. This approach has the advantage of avoiding the massive interaction that exists in the first approach between the two models. In our study we propose an approach to integrate the interoperability security policies in the model of a system. We use extended finite automata to model the functional aspect of a system. The key idea of this approach is that a security policy restrains the functional behavior of the system in which it applies. In some cases the integration of some rules, i.e. obligation rules, adds new behaviors to this system. Such new behaviors have to be properly created and integrated in the model. In our study, we consider dynamic security policies. Thus, the activation of each security rule depends on contextual conditions. In addition, we consider different modalities of security rules such as permission, prohibition and obligation. Finally, by applying our approach, it will be possible to revise the integrated security policies at any time. This is an important feature when considering dynamic creation and destruction of security rules.

Next, we provided a methodology to verify whether the behavior of a model describing a system respects some interoperability security policies. Most of the previous work in this domain focus on the verification of security properties. In these studies, the security policy is modeled without the functional aspect of the system on which it applies. In other work, these models are used to check the correctness of this model with respect to the security policies specifications. However, when we consider dynamic security polices, the activation of each rule is constrained with contexts. In general, a security rule context involves the functional aspect of the system on which it is applied. Thus, modeling only the security policy is not sufficient anymore. This model has to include both the functional aspect and its security policies. In this case, the existing methods cannot be applied anymore as it will not be able to verify if the system model respect the security policy according to a context. In our study, we propose a model checking based approach to verify the correctness of a system model behavior with respect to interoperability security policies. Indeed, in order to applying model checking techniques, interoperability security rules need to be described using formal languages. These security rules will constitute the properties that has to be respected in all the model execution. We use Linear Temporal Logic (LTL) to formally describe interoperability security rules. Moreover, we describe each security rule with two LTL formulae. To verify the correctness of a model, we check whether all the execution of this model respect these LTL formulae.

Finally, we developed a framework for testing whether the behavior of systems respect its interoperability security policies. In particular, we are interested to these behaviors that happen when these system interact with each others. This framework is based on the combination of both active and passive techniques. This framework provide a testing process that is able to check whether the behavior of a system, which receives requests from other ones, respects its security policies according to a specific context. An additional contribution is a hospitals network case study, which is presented in this Thesis to show the feasibility of our proposals.

1.3 Outline of the Dissertation

The rest of the Thesis is organized as follows: Chapter 2 and 3 present the state of the art related to security policies and formal testing respectively. Chapter 4 proposes a formal method for modeling the functional aspect of a system and its interoperability security policies. In particular, we shall present how these policies are integrated in a system model. In Chapter 5, we shall present a framework for verifying the correctness of a system model with respect to its security policy. Chapter 6 focuses on the formal testing process, which checks whether a system respect its interoperability security policies when it is accessed by other systems. Moreover, this chapter presents a testing framework that combines the active and passive testing approaches. Finally, in Chapter 7 we shall present the conclusions and perspectives.

Part I

State of the Art

"Books serve to show a man that those original thoughts of his aren't very new at all."
Abraham Lincoln

Chapter 2

Security Policy Background

Contents

2.1 Introduction

The interaction between business communities becomes a crucial requirement due to the need of exchanging and sharing resources and services. This requirement cannot come without the associated security risks. In order to guarantee information security, the Confidentiality, Integrity and Avalability (CIA) properties have to be preserved.

A security policy is a set of rules that defines the desired behaviour of a user within an organization. Accordingly, each action performed by a user on the organization resources is administrated by the system security policy. Computers, printers, card-readers, sensors, digital photocopiers, databases web services etc., are typical examples of resources. The objective of a security policy is to preserve one or more of the information security properties of an organization. Many works have basically focused on the definition of models in order to specify the security policies such as an access control. The choice between the models can be based on the size of the organization or the security property to enforce (confidentiality, integrity, availability). For instance, a small organization might be satisfied with an access control framework of type access matrix [Lam71] or discretionary access control (DAC) [FJFD01]. The MAC model [FJFD01] which has been closely associated with multi-level secure (MLS) systems is used generally by military organization to protect

critical information. It is defined by the orange book as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity". In commercial environments with large number of users and resources (e.g., hospitals), role-based access control (RBAC) [FK92] may be more suitable to use since it is easier to manage a large number of users in RBAC then other models such MAC or DAC.

Since then, many models have been proposed based on one of these basic models. Each of the proposed models can respond to specific needs but not sufficiently generic to express the different needs that might arise in the security policy of an organization. Specification of a security policy is actually no longer restricted to permissions. A security policy may include three different types of security rule: permission, prohibition and obligation. Intuitively, a prohibition is a negative permission implying that one must not perform some action. An obligation is associated with an action someone must perform and is usually triggered when some conditions are satisfied. Also, the increasing complexity in the requirements of secured interactions between systems urges security models to handle more flexible and dynamic security policies [CCB08]: security rules in these policies are no longer static but dynamic. In a dynamic security policy, the activation of a security rule depends on contextual conditions. For instance, the location of a user can activate a specific set of security rules defining his/her right within this geographical location. The advantage of such dynamicity is to provide a self-adaptive access control security policy that is aware of changes in the interoperability environment. More recently, some models to specify dynamic security policies have been proposed [KBM+03, CCBS05].

In this chapter, we present the background of security policy models. We also present some security specification languages.

2.2 Basic Concepts

Information security is achieved by implementing security policies and procedures as well as physical and technical measures that deliver the following three security properties:

- Confidentiality: prevention of the authorized disclosure of information. The confidentiality represents the assurance that information is shared only among authorized persons or organizations. Breaches of confidentiality can occur when data is not handled in a manner adequate to safeguard the confidentiality of the information concerned. Such disclosure can take place by word of mouth, by printing, copying, e-mailing or creating documents and other data etc. The classification of the information should determine its confidentiality and hence the appropriate safeguard.
- Integrity: prevention of the unauthorized modification of information. The integrity represents the assurance that the information is authentic and complete, and that it can be relied upon to be sufficiently accurate for its purpose.
- Availability: prevention of the unauthorized withholding of information or resources. The availability represents the assurance that the systems responsible for delivering,

storing and processing information are accessible when needed, by those who need them.

Additional basic security concepts not regarded as primitive security properties are:

- Accountability involves the concepts of answerability, responsibility, blameworthiness, liability and other terms associated with the expectation of account-giving.
- Non-repudiation is an information technology service that provides proof that an action took place, can be verified by a third party with high assurance, and cannot subsequently be refuted. It is often applied to authentication or approval but may also involve information integrity, origin, submission, sending, transport, receipt, delivery and knowledge.

A security policy is a set of rules that regulates the nature and the context of actions that can be performed within a system. It permits to govern the choices in behaviour of a system [Slo94]. In practice, organizations define security policies to specify how subjects can access to their information (data and services). Basically, they regulate the information knowledge to preserve the confidentiality, information changes and modifications (to preserve the integrity) and resource availability (to preserve the availability).

In general a security policy can be classified into two types:

- **Authorization policy** defines what activities a subject is permitted to perform on a target object of a system. In general an authorization policy may be positive (permitting) or negative (prohibiting), i.e. not permitted = prohibited.
- **Obligation policy** defines what activities a subject must do. It is an event-triggered condition-action rules that can be used to define adaptable management action. These policies thus define the conditions for performing a wide range of management actions (for example change Quality of service, activation or deactivation user account in a system or install new software, etc.).

A security policy is specified by giving a condition on sets of predefined executions of a system. Accordingly, rather than changing the functionality of the actual operations of a system , security policies offer to condition the behaviour of the system when predefined operations or actions are invoked by the users. A constraint (or security rule context) can optionally be defined as part of a policy specification to restrict the applicability of the policy. It is defined as a predicate referring to global attributes such as time, action parameters or users events.

In large-scale system containing a large number of users and resources, it is not practical to define security policies related to individual entities. It must be possible to specify security policies related to groups of entities. For instance users can be grouped according to the roles or the positions that these users have in an organization, resources can be grouped according to its type.

A Security Policy is derived from a comprehensive risk analysis. The risk analysis involves identifying the assets and the threats that exist to those assets. This analysis may take into account several parameters that include business goals, service level agreements or trust relationships within or between organizations. In this work we consider that an or-

ganization may have two types of access control security policies. *The local security policy* of an organization permits only for authorized users of this organization to access services and resources. It limits the activity of legitimate users of an organization who have been successfully authenticated. *The interoperability security policy* permits to manage interoperability between components having their own policies defined by different organizations. In such case, privileges of a user of an organization who is accessing resources of other organizations can be defined after a negotiation and exchange of credentials [MTCB05] between the organizations.

2.3 Security Policy Specification

Access control policy is the set of security functionalities that define how subjects and objects interact with each other. It reflects the strategy of an organization in terms of information security. Basically, it regulates the information knowledge to preserve the confidentiality, information changes and modifications to preserve the integrity and resource availability to preserve the availability. The basic models for control access are introduced in the following as well as the OrBAC model which will be used later in our work.

2.3.1 Basic Security Policy Models

The study of access control has identified a number of useful access control models, which provide a representation of security policies and allow the proof of properties on an access control system. Access control policies have been traditionally divided into the following three categories.

2.3.1.1 Mandatory Access Control (MAC)

Mandatory access control (MAC) [FJFD01] refers to a type of control access by which the system constrains the ability of a subject to access or generally to perform actions on an object or target. For each object is given a level of sensitivity and for each subject a level of clearance. Thus, a subject will only be allowed to access the object that has a sensitivity level which is equal or lower that his clearance level.

With mandatory access control, the security policy is centrally controlled by a central authority (security policy administrator). Users do not have the ability to override a security rule and, for example, to grant access to files that otherwise would be restricted.

A well known model based on MAC is defined in the 70th by Bell and Lapadula. This model preserves the confidentiality by using the follow two rules:

- The Simple Security Property: a subject at a given security level may not read an object at a higher security level
- The *-property: a subject at a given security level must not write to any object at a lower security level (no write-down).

The *-property, also known as the Confinement property, is to prevent illegal copy of critical information (by using for example an attack of type Trojan horse) to a lower level of sensitivity where a malicious user can access. However it is clear that models based on this model cannot ensure the integrity of the information. A user with a low level of clearance can always corrupt information of higher level of sensitivity.

To preserve the integrity of the information Biba has proposed another model which is based on the following two rules:

- The Simple security property: states that a subject at a given level of integrity must not read an object at a lower integrity level (no read down).
- The *-property: states that a subject at a given level of integrity must not write to any object at a higher level of integrity (no write up).

Other models based on MAC are also proposed, each of these models responds to a specific need. For instance, Clark and Wilson have proposed a model that ensures the integrity for commercial applications, whereas, Muraille proposed a model that preserves the confidentiality in commercial applications.

2.3.1.2 Discretionary Access Control (DAC)

A discretionary access control (DAC) [FJFD01] policy regulates access to objects based on the identity of the subject and/or the group to which they belong. The controls are discretionary in the sense that a subject can pass his/her access permissions to another subject. The notion of delegation of access rights is thus important part of any system supporting DAC. Basic definitions of DAC policies use the access matrix model as a framework for reasoning about the permitted accesses. In the access matrix model the state of the system is defined by a triple (S, O, A), where S is a set of subjects, O is a set of objects and A is the access matrix where rows correspond to subjects, columns correspond to objects and entry $A[s, o]$ reports the privileges of s on o.

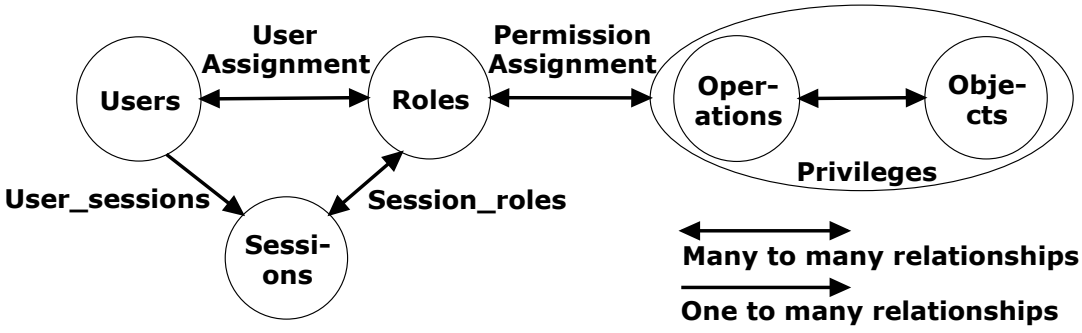


FIGURE 2.1: RBAC Model.

2.3.1.3 Role based access control (RBAC)

A Role based access (RBAC) [FK92] model is more generic then the two previous models. The rules are associated to a set of roles which are assigned to subjects. Thus, since

users are not associated to permission directly, but only acquire them thorough their roles, management of the security policy will be easier than a MAC or a DAC policy based. Operations such as adding users or changing users or roles are simplified.

Within an organization, roles are created for various job functions. The permission to perform certain operations is assigned to specific roles. System users are assigned to particular roles, and through those role assignments acquire the permission to perform particular system functions. Roles permit the grouping of a set of permissions related to a position in an organization. A number of models based on RBAC have been proposed. Figure 2.1 describes the first proposed model, namely $RBAC_0$, based on RBAC. In this model we can distinguish three deferent relations: a relation that links a user to a role, a relation that links a role to permissions and a relation that links a session to a pair (user, role). Thus, when a user authenticate, he will activate some of the roles to which he is assigned. As long as the session is open this user can thus have all the accesses which are given to the activated roles. After $RBAC_0$, three other models have been proposed: $RBAC_1$, $RBAC_2$ et $RBAC_3$. $RBAC_1$ model adds to the original model the concept of role hierarchy, $RBAC_2$ adds the notion of constraints. The model $RBAC_3$ includes the notions defined in $RBAC_1$ and $RBAC_2$

2.3.2 OrBAC Model

The Organization Based Access Control (OrBAC) [KBM+03] model is an access and usage control model. The concept of organization is central in OrBAC. Intuitively, an organization is any entity that is responsible for managing a security policy. For this purpose, OrBAC defines an organizational level that is abstracted from the implementation of this policy.

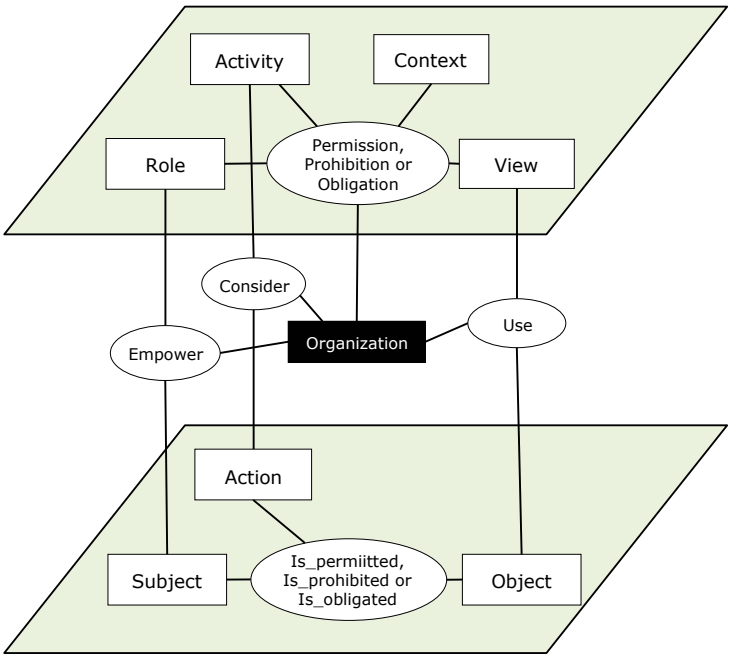


FIGURE 2.2: OrBAC Abstraction Layers.

An organization can use OrBAC to specify its own security policy at the organizational level. Thus, instead of modeling the policy by using the concrete and implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned in the organization. The role of a subject is simply called a role as in the RBAC model. On the other hand, the role of an action is called an activity whereas the role of an object is called a view. A security rule is thus described in OrBAC as a role having the permission, prohibition or obligation to perform an activity on a view in a given context.

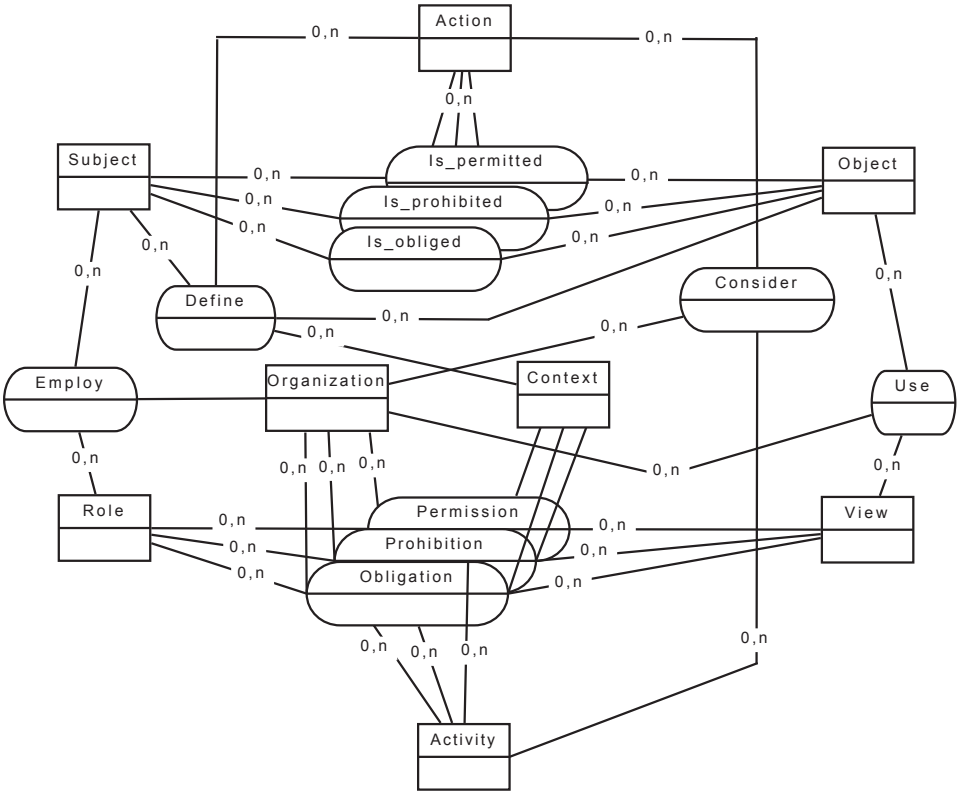


FIGURE 2.3: OrBAC Model.

Figure 2.2 and 2.3 illustrate respectively the two abstraction levels and the relationships between the different entities of the OrBAC model.

Definition 1 *A security rule is a relation between organization roles, views, activities and contexts. It is defined as a role having permission, prohibition or obligation to perform an activity on a view within an organization.*

Accordingly, a security rule in OrBAC has the following form:

$SecurityRule(org, modality(role, activity, view, context)$

where *modality* belongs to {permission, prohibition, obligation}. In the case where the *modality* is a permission, the security rule means that in organization *org* a *role* has the permission to perform *activity* targeting the resource represented in *view* if *context* is

satisfied. Once the organizational security policy has been defined, one must test how this policy is applied to concrete entities that are the subjects, actions and objects. To assign concrete entities to abstract entities, OrBAC uses three predicates:

- $Empower(org, subject, role)$: means that in the organization org, the subject $subject$ is assigned to the role $role$.
- $Consider(org, action, activity)$: means that in the organization org, $action$ is considered as an implementation of $activity$.
- $Use(org, object, view)$: means that in the organization org, $object$ is used in $view$.

2.3.2.1 OrBAC: The Context

The increasing complexity in the requirements of secured interactions between systems urges security models to handle more flexible and dynamic security policies: the activation of a security rule depends on contextual conditions.

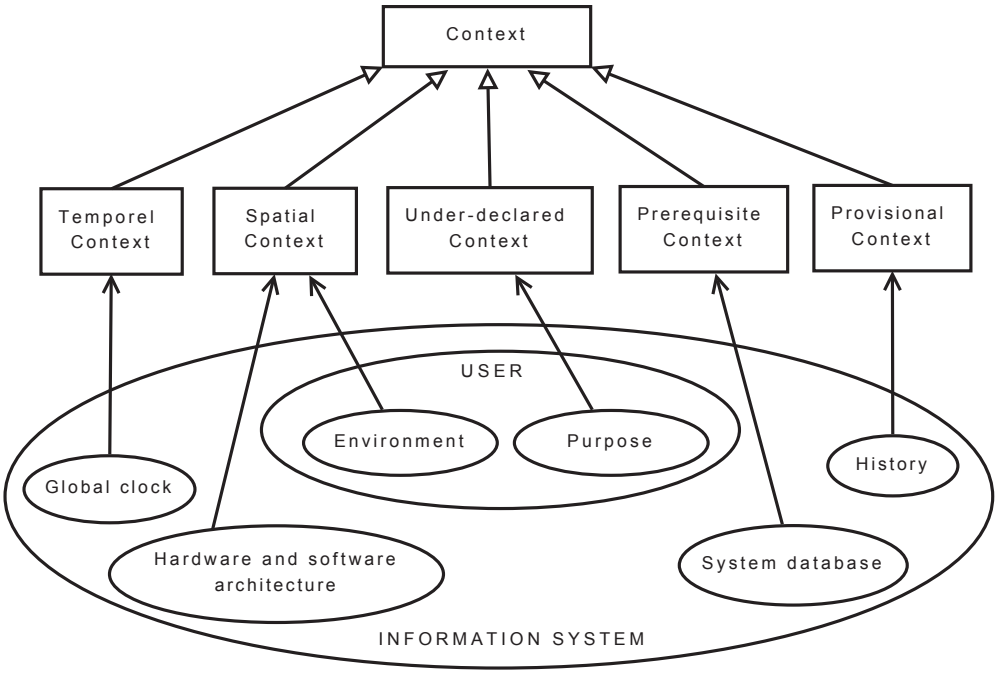


FIGURE 2.4: OrBAC Context.

In OrBAC, we use context [CCB08] to express different types of extra conditions or constraints that control activation of rules expressed in the access control policy:

- The temporal context that depends on the time at which the subject is requesting for an access to the system.
- The special context that depends on the subject location.
- The user-declared context that depends on the subject objective (or purpose)

- The prerequisite context that depends on the characteristics that join the subject, the action and the object.
- The provisional context that depends on previous actions the subject has performed in the system.

OrBAC assumes that each organization manages some information system that stores and manages some different types of information. To control activation, each information system must provide the information required to check that conditions associated with the definition are satisfied or not. The following list gives the kind of information related to the contexts we have just mentioned.

- A global clock to check the temporal context.
- The subject environment and the software and the hardware architecture to check the special context.
- The subject purpose to check the user-declared context.
- The system database to check the prerequisite context.
- A history of action carried out to check the provisional context.

In figure 2.4, we illustrate the correspondence between the contexts and the required data. In the case where the system does not provide the needed information, then the corresponding context cannot be managed by the access control policy.

2.3.2.2 The O2O Approach

OrBAC model supports the O2O approach [CCBCB06] to manage interoperability between components having their own policies defined by different organizations. The O2O approach relies on the concept of virtual private organization (VPO) to designate the sub-organization in charge of the interoperability access control.

To explain the basic principles of O2O, let us consider that a given organization A that wants to interoperate with another organization B. In this case, each organization has to define a Virtual Private Organization (VPO) respectively called A2B and B2A. The VPO A2B is associated with a security policy that manages how subjects from the grantee organization A, O-grantee, may have an access to the grantor organization B, O-grantor. We say that the VPO A2B manages the interoperability security policy from organization A to organization B. The VPO B2A is similarly defined to control accesses of subjects from organization B to organization A. In O2O, the concepts of authority sphere and management sphere specifying who creates and manages the interoperability security policy are introduced. Figure 2.5 illustrates the O2O framework.

The following three entities are considered in O2O:

- O-grantor: the organization that hosts resources to be accessed;
- O-grantee: the organization that requests access to the resources of O-grantor;
- Virtual Private Organization (VPO): a VPO is associated with two attributes, which are O-grantor and O-grantee.

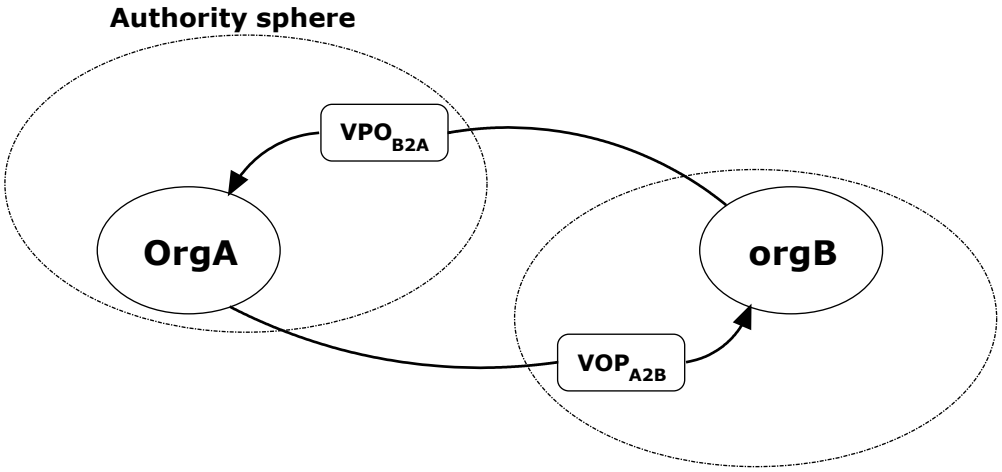


FIGURE 2.5: The O2O Approach.

The O-grantor attribute represents the organization which has created the VPO in order to grant access to subjects on resources from another organization represented by the O-grantee attribute. Thus, in a VPO, the O-grantor organization can define roles, activities and views and associate these roles, activities and views with contextual security rules as in a classical organization of the OrBAC model. When assigning subjects, actions and objects to the respective roles, activities and views defined in the VPO, the following restrictions apply:

- If a given role is assigned to a given subject in a VPO, then this subject must come from the O-grantee organization of this VPO. This is because a VPO is designed to control how subjects from the O-grantee organization may have an access to the O-grantor organization.
- If a given object is used in a given view in a VPO, then this object must be used in some view in the O-grantor organization. This is because, in a VPO, the O-grantor organization can only grant access to its "own" objects.
- If a given action is considered an implementation of an activity, then this action must be controlled by the O-grantor organization. This will be the case when this action is directly implemented by the O-grantor organization.

2.3.3 Other Security Policy Models

Other security policy models are proposed in the literature. It is very difficult to cover all of them for matter of space. We can quote for example the Non Atomic Action and Deadlines (Nomad) model. The main advantages of Nomad model are to provide means to specify: (i) privileges that is permission, prohibition or obligation, associated with non atomic actions, (ii) conditional privileges which are privileges only triggered when specific conditions are satisfied and (iii) privileges that must be fulfilled before some specific deadlines. Also several models based on RBAC have been proposed to include constraints, the model TRBAC [BBF01] which takes into consideration the aspect of time in the activation and the deactivation of roles. GTRBAC [JBLG05] is a generalization of TRBAC, it takes into

consideration each relation defined in the RBAC model and add time constraints to those relations. Thus, we can constraint the activation of a role, the assignment of a user to a role, the assignment of a role to permissions. The model GeoRBAC [BCDP05] is also based on RBAC, it includes the geographical location of a user as a constraint to satisfy in order to activate roles. We can also quote the Open Distributed Programming reference model (ODP-RM), and some extensions of OrBAC model like PolyOrBAC [Des11]. In [TCEM12] we provide a comparative study between OrBAC and RBAC security policy models for distributed systems.

2.3.4 Security Policy Specification Languages

2.3.4.1 Logic-Based Languages

Logic-based languages have been widely used as a logical framework that provides a means of specification of security policies. The key advantage is that proof-based validation of some properties of security policy can be achieved by using logic-based languages as formalism. The OrBAC language is based on a first order logic to represent the relationships between the model entities described previously in figure 2.3. In the other hand, Nomad language combines deontic and temporal logics to allow the description of conditional privileges and obligation with deadlines. Moreover, many other work has focused on describing RBAC policies in different logic-based languages. These include the logical notation introduced in [CS96] and the Role Definition Language presented in [HBM98]. In addition, the work done in [Bos95] has applied the formal specification language Z to specify and validate the security model for the NATO Air Command and Control System. The aim of this work was to develop a model for both mandatory and discretionary access controls based on the Bell-LaPadula approach mentioned previously.

Although there is some works that adopted the deontic language for security policy specification. In [CC97], the deontic language is used to represent a security policy with the aim of detecting conflicts in security policy. Another interesting approach is the Barker model [Bar] that adopts an approach to express range of access control policies using satisfied clause form logic, with emphasis on RBAC policies.

2.3.4.2 XACML

The eXtensible Access Control Markup Language (XACML) is an OASIS standard [XAC11] which describes both a policy language and an access control decision request/response language. Both of those languages are written in XML. The policy language is used to describe general access control requirements to resources in an information system. The request/response language allows us to form a query to ask whether or not a given action should be allowed and the response will convey the answer for this query. The response always includes an answer about whether the request should be allowed using one of four values: Permit (access allowed), Deny (access denied), Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (this service has no policies that apply to this request).

Similar to existing policy languages, XACML is used to specify subject-target-action-condition oriented policy in the context of a particular XML document. It is a powerful extensible language. The standard language already supports a wide variety of data types, functions, and rules about combining the results of different policies. In addition to this, there are already standards groups working on extensions and profiles that will hook XACML into other standards like SAML and LDAP, which will increase the number of ways that XACML can be used.

2.3.4.3 SPL

The security policy language (SPL) [RZFG99] is an event-driven policy that supports access-control, history based and obligation-based policies. SPL is implemented by an event monitor that, for each event, decides whether to allow or ignore the queried to determine the subject who initiated the event, the target on which the event is called, and attribute values of the subject, target and the event itself.

2.3.4.4 PDL

The policy description language (PDL) [LBN99] is an event-based language from Bell-labs in which they used the event-condition-action rule paradigm of active databases to define a policy as a function that maps a series of events into a set of actions. The language can be described as a real-time specialized production rule system to define policies. The syntax of PDL is simple and policies are described by a collection of two types of expressions: policy rules and policy defined event propositions. Policy rules are expression of the form: *event causes action if condition*, which reads: if the event occurs under the condition the action is executed. Policy defined event propositions are expressions of the form: *event triggers policy-defined-event if condition*, which reads: if the event occurs under the condition, the policy-defined-event is triggered.

2.3.4.5 TPL

The trusted policy language (TPL) by IBM [HMM+00] provides a clearer separation between the authentication of subjects based on certificates and the assignment of authorization of subjects based on certificates and the assignment of authorizations to those subjects which have been successfully authenticated. With TPL, the credentials result in a client assigned to a role which specifies what the client is permitted to do, where a role is a group of entities that can represent specific organizational units (e.g. employees, managers, auditors). The assignment of access right to roles is outside the scope of TPL; the philosophy of the work on TPL is to extend role-based access control mechanisms by unknown users to well defined roles. There also exist other languages for security policy specification; we can quote for example Ponder language which is a Policy Language for Distributed Systems Management, Role definition language (RDL) and LaSCO which is a graphical approach for specifying security policies.

2.4 Conclusion

In this chapter we presented a global view on the existing models and languages for access control security policy specification. This step helped us to identify the differences between those models and thus to choose among them the most suitable for our application. In our work, we are interested in interoperability security policies. In order to specify an interoperability security policy we need a model able to specify dynamic access control security policies with different type of privileges and able to manage interoperability between components having their own policies defined by different organizations. A dynamic access control security policy means that the activation of a security rule of this policy can be constrained by a specific condition called the context of the security rule. We found that OrBAC model is a good candidate: i) it can define three different types of privileges (which are permission, prohibition and obligation); ii) The security rules are specified in an organizational level independently from its implementation; iii) it uses contexts to express different types of extra conditions or constraints that control activation of rules expressed in the access control policy; iv) OrBAC model supports the O2O approach to manage interoperability between components having their own policies defined by different organizations. The O2O approach relies on the concept of virtual private organization (VPO) to designate the sub-organization in charge of the interoperability access control.

It is also necessary to define a formal logic based representation of the security policies in order to support the verification process. Based on these formal specifications, it will be possible for us to investigate in an easier way many research issues related to the interoperability security policy verification and testing. Indeed, to check whether a system correctly implement its security policy, we can rely on these specifications combined with interoperability testing techniques to either automatically derive a set of complete test scenarios targeting security objectives or to monitor a running system to check it conformance with respect to the security requirements. In the next chapter the existing formal techniques for verification and testing are presented.

"A journey of a thousand miles begins with a single step."
Confucius

Chapter 3

Formal Testing

Contents

3.1 Introduction

It is straightforward to see that system that deals with security aspects must be tested before they are available for being used. The best way to guarantee the correctness of a system is to apply formal methods.

Formal methods are based on rigorous mathematical concepts used to describe and analyze the system behaviour. The main advantage of using formal methods is the aptitude to automatically test if the behaviour of a system verifies some properties based on dedicated tools. The disadvantage is that these languages are not usually easily used by an ordinary user without an adequate training.

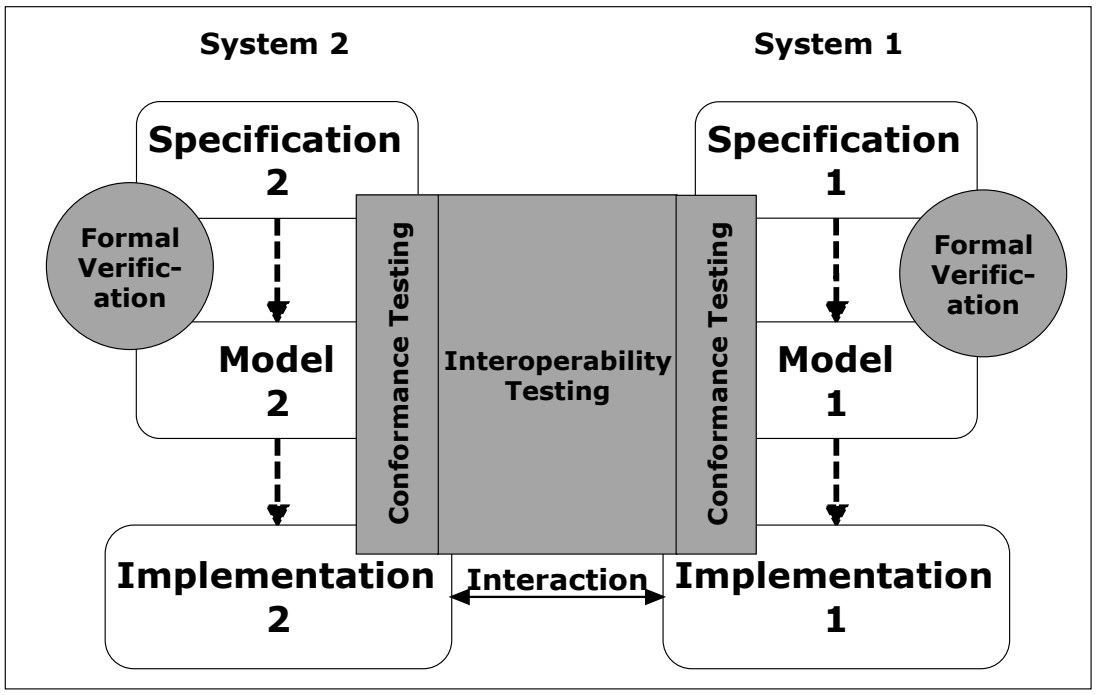


FIGURE 3.1: Formal Testing Overview.

Based on formal methods, system development can be decomposed into four phases which are presented in Figure 3.1:

- *The system behaviour specification*, i.e. how the system is supposed to behave is modelled using formal models called also system specifications. Specification is, therefore, the phase where an abstract model of the system is produced.
- *Verification* consists in checking that the system specification has no faults. For instance, it verifies that the system specification respects some system properties extracted from the system requirements.
- In the *implementation phase*, the system becomes real. From this phase we do not deal with any abstract model, but a set of programs that react with environment or to other systems and/or users.
- Finally, *testing* is the process of checking that the implementation meets its specification and it fulfils its intended purpose.

In the case of communicating systems, two types of testing is to be considered: conformance testing and interoperability testing.

The conformance testing checks whether an implementation conforms to its specification. Therefore, conformance improves the chances of interoperability of systems which are based on the same specification, but it does not prove end-to-end functionality between communicating systems. Interoperability is defined as the ability of exchanging and using information between two or more systems (or components) [ETS07].

3.2 System Modeling

In this section we present some formal models existing in literature. In particular, these models allow us to describe the behaviour of collections of parallel processes that may compute independently or interact with each others. It provide us with facilities for the description of well-known phenomena that appear in the presence of concurrency and are familiar to us from the world of operating systems and parallel computation in general (e.g., deadlock, livelock, starvation and so on).

Our aim in the remainder of this section will be to give a general description on some modelling techniques that can be used to describe, and reason about any collection of interacting processes. The automaton approach will be presented in more details as it will be used in this work for modelling and testing communicating systems. Those approaches include:

- Finite automata (labeled transition systems),
- process algebra and
- Petri Nets.

3.2.1 Finite Automata

In Finite Automata (FA) model, processes are represented by the vertices of certain edge-labelled directed graphs and a change in process state caused by performing an action is understood as moving along an edge, labelled by the action name, that goes out of that state. A FA system consists therefore of a set of states, a set of labels (or actions) and a transition relation T describing changes in process states: if a process p can perform at a state s_1 an action a and goes to a state s_2, we write $s_1 \xrightarrow{a} s_2$.

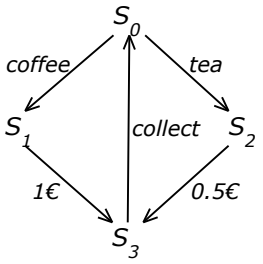
Example 1 *Let us consider the classical example of tea and coffee machine. The behaviour of this machine can be described as follows. From the initial state, let us say s_0 representing the situation waiting for a request, two possible actions are allowed. A consumer can order a coffee or a tee. After pressing the corresponding button the internal state of the machine goes to the state s_1 or s_2 representing respectively that the user has ordered a coffee or a tee. Formally, this can be described with the following two transitions. Accordingly, the state s_1 represents that the consumer has ordered a coffee whereas the state s_2 specifies that the consumer has ordered a tee. Formally, this can be described by the transitions*

$$s_0 \xrightarrow{coffee} s_1 \text{ and } s_0 \xrightarrow{tea} s_2.$$

Now the consumer will be asked to insert a specific amount of money in order to get the order. After inserting the correct amount of money the internal state of the machine goes to a state. This state represents that the machine has received the payment for the chosen drink. These state changes can be modelled by the transitions

$$s_1 \xrightarrow{1€} s_3 \text{ and } s_2 \xrightarrow{0.5€} s_3.$$

Finally the drink is collected and the machine goes to its initial state. The above machine can be represented with the following FA:



Definition 2 *A labelled transition system* (S, s_0, L, F) *consists of*

- *a set* S *of states*
- *an initial state* $s_0 \in S$
- *a set* L *of action labels*
- *a transition relation* $T \subseteq (S \times L \times S)$.
- *an accepting run of a FA is a finite run in which the final state is in* F.

A labelled transition system is finite if its sets of states and actions are both finite.

3.2.1.1 Extended Finite Automata

An Extended Finite Automata (EFA) is an augmentation of the ordinary automata with guard functions (predicates) and action functions. We consider that a transition can be executed only when an input is received and the predicate is true.

Definition 1 *An extended automaton is a 5-tuple*

$$E = \langle Q, \vec{v}, T, \Sigma, q_0 \rangle$$

where:

(i) Q *is a finite set of states and* q_0 *is the initial state;*

(ii) $\vec{v} = (v_1, ..., v_n)$ *is a vector of typed variables;*

(iii) $\Sigma \subseteq (L_i \times L_o)^*$ *is a nonempty set of input/output alphabet where* L_i *and* L_o *are the set of inputs and outputs respectively.*

(vi) T *is a set of transitions, defined by a tuple* $\langle q, \sigma, \wp, a, q' \rangle$*, where*

(i) q *and* q' *is the source and the target state respectively;*

(ii) $\sigma \in \Sigma$ *is the input/output of the transition;*

(iii) a *is an action of the transition. The action updates the values of* $\vec{v}$ *after execution of the transition;*

(iv) $\wp$ *is a predicate over the values of* $\vec{v}$*. The transition is activated only if* $\wp$ *is true.*

In our work we used the extended automata in order to specify the functional behaviour of communicating systems. The interaction between these systems is assumed administrated with interoperability security policies. In fact, a security policy restricts the normal behaviour of a system for a specific user in order to regulate its actions on the system. This can be described as disabling or enabling the execution of some transitions by the usage of the transition predicate. Also it is possible that the security policy introduces some new behaviours to the functional model of the system. We show that those new behaviours can be described using this formalism. More details will be given in Chapter 4.

3.2.1.2 Timed Automata

A timed automaton is a standard finite-state automaton extended with a finite collection of real-valued clocks. The transitions of a timed automaton are labelled with a guard (a condition on clocks), an action, and a clock reset (a subset of clocks to be reset). Intuitively, a timed automaton starts execution with all clocks set to zero. Clocks increase uniformly with time while the automaton is within a node. A transition can be taken if the clocks fulfill the guard. By taking the transition, all clocks in the clock reset will be set to zero, while the remaining keep their values. Thus transitions occur instantaneously. Semantically, a state of an automaton is a pair of a control node and a clock assignment, i.e. the current setting of the clocks. Transitions in the semantic interpretation are either labelled with an action (if it is an instantaneous switch from the current node to another) or a positive real number i.e. a time delay (if the automaton stays within a node letting time pass).

3.2.2 Petri Nets

A Petri net consists of places, transitions, and directed arcs. Arcs run from a place to a transition or vice versa, never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places may contain a natural number of tokens. A distribution of tokens over the places of a net is called a marking. A transition of a Petri net may fire whenever there is a token at the start of all input arcs; when it fires, it consumes these tokens, and places tokens at the end of all output arcs. A firing is atomic, i.e., a single non-interruptible step. Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire. If a transition is enabled, it may fire, but it does not have to. Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems.

Definition 3 *A Petri net graph (called Petri net by some, but see below) is a 3-tuple* (S, T, W)*, where*

- *S is a finite set of places*
- *T is a finite set of transitions*
- *$W : (S \times T) \cup (T \times S) \to \mathbb{N}_W$ is a multiset of arcs, i.e. it defines arcs and assigns to each arc a non-negative integer arc multiplicity; note that no arc may connect two places or two transitions.*

3.2.3 Process Algebra

The term *process algebra* was defined in 1982 by Bergstra & Klop [BK82]. The word *process* here refers to behavior of a system. Behaviour is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, we describe certain aspects of behaviour, disregarding other aspects, so we are considering an abstraction or idealization of the *real* behaviour. Rather, we can say that we have an observation of behaviour, and an action is the chosen unit of observation. Usually, the actions are thought to be discrete: occurrence is at some moment in time, and different actions are separated in time. This is why a process is sometimes also called a discrete event system.

The word *algebra* denotes that we take an algebraic/axiomatic approach in talking about behaviour. That is, we use the methods and techniques of universal algebra .

When talking about process algebra, we usually consider it as an approach for high-level description of interactions, communications and synchronization between a collection of agents and processes. Thus, we can say that process algebra is the study of the behaviour of parallel or distributed systems by algebraic means. It offers means to describe or specify such systems, and thus it has means to talk about parallel composition. Besides this, it can usually also talk about alternative composition (choice) and sequential composition (sequencing). Moreover, we can reason about such systems using algebra, i.e. equational reasoning. By means of this equational reasoning, we can do verification, i.e. we can establish that a system satisfies a certain property. Leading examples of process calculi include CSP(communicating sequential process), CCS (calculus of communicating system), ACP (algebra of communicating process), LOTOS, π-calculus and ambient calculus.

3.3 Formal Verification

Formal verification is the process of checking whether a design satisfies some requirements (properties). To verify a system we need to describe two things: the set of properties we want to verify, and the relevant aspect of the system related to those properties.

Figure 3.2 illustrates the formal verification process. Two well-established approaches to verification are model checking and theorem proving.

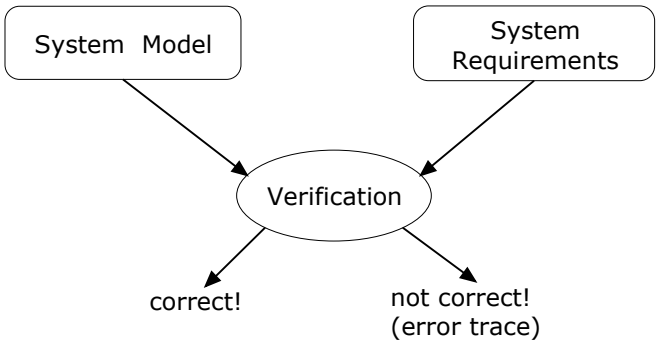


FIGURE 3.2: Formal Verification.

3.3.1 Model Checking

Model checking is a technique that relies on building a finite model of a system and checking that desired properties hold in this model. Roughly speaking the check is performed as an exhaustive state space search which is guaranteed to terminate since the model is finite.

Two main approaches for model checking are used today. The first is the temporal model checking which is developed in the 1980s by Clarke and Emerson [CE81]. In this approach the properties to be checked are represented in temporal logic and the systems are represented in finite state transition systems. An efficient search is used to check whether the properties are respected in the system model. In the second approach, the properties are given as an automaton; then the system, also modeled as an automaton, is compared to the properties to check whether or not these properties are respected by the system model.

The main advantages of model checking technique are: (1) it is completely automatic and fast, (2) model checking can be used to check partial specifications, and thus it can provide useful information about system correctness even if the system has not been completely specified and (3) it produces counterexamples which represents subtle errors in design, and thus it can be useful for debugging. On the other hand, the main disadvantage of model checking is the state space explosion problem that can happen when performing exhaustive state space search of some large systems.

3.3.2 Theorem Proving

Theorem proving [EFN71] is another method for performing verification on formal specifications of system models. Theorem provers apply inference rules to a specification in order to derive new properties of interest. The theorem proving tools consist of a powerful collection of inference steps that can be used to reduce a proof goal to simpler sub-goals that can be discharged automatically by the primitive proof steps of the prover.

Given a property and a model, the user is either able to verify the property by completing the proof or else given back un-dismissed subgoals that give scenarios in which the property is violated. Theorem proving is generally harder then model checking. It requires considerable technical expertise and understanding of the specification. But it gives the user a lot more flexibility and control in doing the proofs. This can give the user greater

insights into the specification. One of the major disadvantages of using theorem provers is that if you fail to complete the proof of a property, the tool will not tell you whether the property is indeed false or whether the user is not providing it with enough information to complete the proof. One of the advantages of theorem provers is that they are not limited by the size of the state space. Large systems that cannot be verified using the model checker can still be verified by the theorem prover. Since state space explosion is not a problem, no abstraction techniques need to be applied and the verification can be done on the complete model. Most theorem provers are highly expressive. Some properties that cannot be easily specified using model checkers (such as comparing properties of two arbitrary states that are not temporally related) can be easily specified in the languages of most theorem provers. These are some of the major advantages of using theorem provers.

3.3.3 Security Policy Verification

In our work, we applied verification technique in order to check the correctness of security policies which are integrated in the functional model of systems. This step is very important in order to give a higher confidence on a secured system model and whether it respects the security policy requirement. In the literature many work have focused on the verification of security policies. These woks can be divided into two main approaches: verification of correctness and verification of security policy properties. While the latter tries to formally model the security policy and then verifies fundamental policy properties, i.e., completeness, termination, consistency and confluence, the former tries to verify the correctness of a security policy which is modeled with the functional model of the system. The common point between these two approaches is that security policies and the properties to be checked have to be properly modeled. This section provides a survey on the latest works in this field.

3.3.3.1 RBAC Security Policy Analysis

Many work have focused on analyzing security policies using Role Based Access Control (RBAC). In all these works, security rules are expressed in a specific formal language that permits the analysis of the security policy. In the work of Drouineaud *et al* [DBTS04], a method to analyze the correctness of security policies using RBAC is proposed. The authorization constraints of the security policy are expressed with a single sorted first order Linear Temporal Logic (LTL). They have provided proofs for some properties of this policy that have been verified by the Isabelle/HOL theorem prover.

Sohr *et al* [SDA05] shows how to formally specify access control policies in clinical information systems. The authors consider first-order linear temporal logic (LTL) as a good formalism for specifying dynamic behaviour of security policies consisting of delegation constraints or constraints which mandate a certain order of task execution as needed for workflows.

The work presented in [SDAG08] discuss two different approaches for security policy specification and verification. The first approach is more formal and uses a theorem prover

for the verification. In contrast, the second one is more practical and is based upon a validation tool for software models. In the formal approach, the security policy is expressed with the LTL formalism. The formal verification assures that an RBAC policy satisfies the properties intended by the organization (e.g., no user may prepare and approve the same check). This verification is carried out by means of the theorem prover Isabelle. The first-order LTL (including axioms) is embedded into Isabelle. For the practical specification and verification, the authors employ the Unified Modeling Language (UML) and Object Constraint Language (OCL). They demonstrate in this paper how to employ the USE system (UML-based Specification Environment) to verify RBAC policies formulated in UML and OCL. The verification is done by generating snapshots (system states) as prototypical instances of a UML/OCL model and compare the generated instances with the specified model (i.e., the RBAC policy in our case).

Another work [FAl09] has developed a framework for the specification of complex RBAC models. It is based on the modularization of the participating access control concepts. Each concept is packed into a so-called authorization module and can then be reused and combined with other modules in order to form an access control model. The framework is designed with the systematic approach of formal object-oriented specification using Object-Z.Multiple inheritance in Object-Z is used in order to modularize access control concepts and explain the main features of the framework. Firstly, the framework can be applied in order to generate a software model for the implementation of an authorization engine supporting a certain combination of access control concepts. Secondly, with the framework new concepts can be defined and explored rapidly and concisely. The authors also demonstrate how to apply formal reasoning in order to provide an indepth analysis of the authorization modules. Thereby they proved important security properties as well as argue about design decisions.

In all the discussed work, security policies are modeled using RBAC model which does not consider contextual security policy. Also security rules are restricted to permission constraints. We consider security rule that includes different modalities of security rules such as permission, prohibition and obligation. Thus the above work has to be extended to cope with our requirements.

3.3.3.2 Huang *et al*

In this work [HK10], the authors introduced a systematic and formal methodology to model security policies and to verify whether required policy properties are assured by the composition of the sub-policies of the systems. It defines fundamental policy properties, i.e., completeness, termination, consistency and confluence, in Petri net terminology and gets some theoretical results.

The security policy is specified with a colored Petri Net which has only one entry place and one exit place. Then they defined each of the security properties based on this security policy specification. This work also provides a study on modular security policy composition. A security policy could be composed of several modules that have to be combined in order to obtain the global policy of a system. This composition usually

produces inconsistent and non-terminating policies. A security policy terminates if the evaluation of every incoming request terminates. The authors explore which syntactic conditions and which operators can guarantee the preservation of these suitable properties for the composition of two policies.

This work lacks details on the specification of the security policy. The authors mentioned the work [ZHL06] that applies Petri Nets to specify Chinese wall policy. It does not provide a generic approach to specify contextual security policy with different modalities such as permission prohibition and obligation.

3.3.3.3 Mallouli *et al*

Mallouli presented in his work [MOC+07] a method for the integration of OrBAC security policies in the functional model of a system. The proposed approach allows the integration of contextual security policies and with different modalities such as permission prohibition and obligation in the functional model of a system.

This method cannot be directly used for the integration of an interoperability security policy. In general, an interoperability security policy is integrated in a system that already contains local security policies. In this case, this method cannot be used directly as it may create security conflict between the existing and the newly added security policy. It has to be extended to support such requirements. Also this method does not support the integration of event based contextual security rules which is needed in our work.

3.4 Conformance Testing

Testing is the process of trying to find errors in a system by means of experimentation. The experimentation is usually carried out in a special environment, where normal and exceptional use are simulated. The aim of testing is to gain confidence that during normal use, the system will work according to its specification. Since testing of realistic systems can never be exhaustive, because testing is time consuming, systems can only be tested during a restricted period of time. The experience and studies have shown that testing may consume up to 50% of the time of the project resources. Thus, testing cannot ensure complete correctness of an implementation. It can only show the presence of errors, not their absence.

In software testing we can distinguish two types of testing: functional testing and structural testing. Structural testing, also referred to as white-box testing, is based on the internal structure of a computer program. The aim is to exercise thoroughly the program code, e.g. by executing each statement at least once. Tests are derived from the program code. In functional testing, externally observed functionalities of a program are tested from its specification. The system is treated as a black box, whose functionalities are determined by observing it, i.e. no reference is made to the internal structure of the program. The main goal is to determine whether the right (with respect to the specification) product has been built. Functional tests are derived from the specification. We also refer to this type

of testing as black box testing.

Conformance testing is a kind of functional testing: an implementation, also referred as System under test (SUT), is solely tested with respect to its specification. Only the observable behaviour of the implementation is tested, i.e. the interactions of an implementation with its environment. In [etsi] the purpose of conformance testing is defined as: *to determine to what extent a single implementation of a particular standard conforms to the individual requirements of that standard.* There exist two approaches for testing the conformance: active testing and passive testing. We will illustrate both of these approaches in the following.

3.4.1 Active Testing

The strength of the active testing is in the ability to focus on particular aspects of the implementation. Indeed the test cases can be for instance limited to a specific type of errors, or to an important state of the system. On the other hand, the test case choice and production can turn out to be complicated. Besides, testing a system can disturb its normal functioning. For this reason, the testing is usually performed on the system implementation before its commercialization.

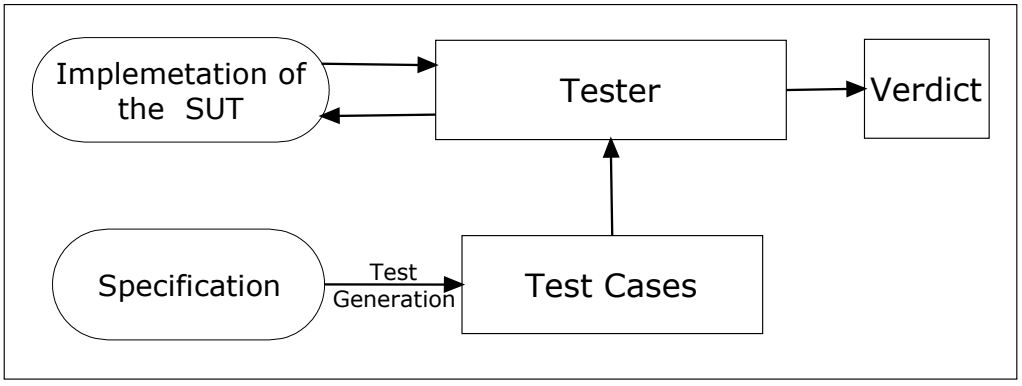


FIGURE 3.3: Model Based Testing Methodology.

The Figure 3.3 describes the steps for the active testing general methodology. Based on the formal specification of the system under test, we generate a set of test cases that we provide to the tester unit. This tester will apply the test cases on the system implementation to stimulate it and collect then its observable behaviour (output messages). These messages are analyzed: the tester checks whether they correspond to the desired outputs described in the specification. If it is the case, the verdict would be *pass*, otherwise, it is *fail*. In some cases (for instance, when the system is non deterministic), the verdict can be *inconclusive*.

3.4.1.1 Automated Test Generation

In practice, the test generation is performed by humans after studying the specification [Tre02]. In formal methods, we build a model that represents the system specification

which allows an automatic test case generation. As illustrated in Figure 3.3, in Active testing technique the system model is an input to generate test suites based on dedicated algorithms. These algorithms allow the process of generating and selecting test cases. In a test generation process, there are two important concepts that must be considered, that are soundness and completeness. We define these two concepts based on [BFS05]:

- soundness: generated test cases should never cause a fail verdict when executed on a correct implementation.
- completeness: for each possible implementation that does not conform to the specification model, it is possible to generate a test case that causes a fail verdict with respect to the system under test.

A number of methods have been proposed to generate test cases from the functional model of the system. One of the strategies to generate test cases is the purpose based test case generation. This strategy has been used to avoid the state space explosion problem. It selects parts of the specification model for being tested, and ideally, the critical properties of the specification that must be validated. The strategy is to generate scenarios that correspond to the given properties. In this context, the test coverage considers a complete test generation if all set of chosen properties are specified in the suit. The model checking technique has been applied to produce model based test cases covering the requirements by using the counter-example provided by the model checker. In this case, the property will become a non desirable property to be checked. In addition to the model checker several dedicated tools has been developed such as TGV [JJ05] and TestGen-if [COML08].

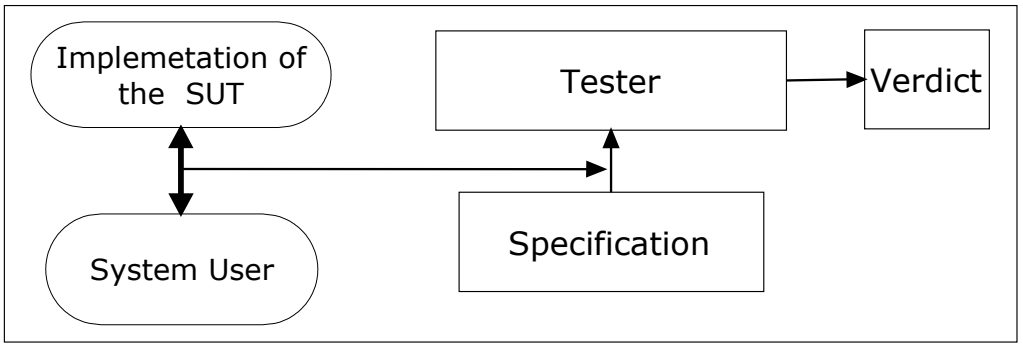


FIGURE 3.4: Passive Testing Methodology.

3.4.2 Passive Testing

Passive testing consists in observing the input and output events of an implementation in run-time. It should not disturb the natural run-time of a protocol or service, that is why it is called passive testing. It is sometimes also referred to as monitoring. The record of the event observation is called an event trace. This event trace will be analyzed according to the specification in order to determine the conformance relation between the implementation and the specification. Figure 3.4 presents the passive testing methodology. We should keep in mind that when an event trace conforms to the specification it does not means that the

whole implementation conforms to the specification. In the other hand, if a trace does not conform to the specification then the implementation neither.

3.4.3 Conformance Testing of Security Policies

We divide the work related to security policy testing into two parts: the approaches based on active testing and the approaches based on passive testing.

3.4.3.1 Active Testing of Security Policies

Some works propose to test the conformance of a security policies with respect to its specifications using active testing techniques. In the following, we discuss some of the latest work in this field.

Mallouli *et al* [MOC+07] have proposed a method to check the conformance of security policy to its specification. They first integrated the security policy in the functional model of a system. The result of the integration step is a new model which is different from the initial one. This model includes the security consideration specified in the security policy. The authors have used a dedicated tool SIRIUS which is a purpose based test generation tool to generate test case from this new model. Their method is applied to generate test cases to test the security policy of a weblog system.

In [SBC05], the authors show that an organization's security policy can be formally specified in a high-level way, and how this specification can be used to automatically generate test cases to test deployed system. By contrast to other firewall testing methodologies such as penetration testing [Cad05], this approach tests conformance to a specified policy, these test cases are organization-specific -i.e. they depend on the security requirement and on the network topology of an organization- and can uncover both in the firewall products themselves and in their configuration. However, this model is limited to the network management and specifically to network and transport layer of the TCP/IP stack. Moreover, it is still a theoretical approach an there exist no tool yet to automate the testing process and evaluate its effectiveness on a real case study.

In [DFG+06] the authors choose another approach to achieve testing of network security policies. They express the network behavior using labeled transition system formulas. Then, for each element of their language and each type of rule, they propose a pattern of test called a *tile*. Then, they combine those tiles into "complete" test cases for the whole rule to perform validation. The test generation method is based on the combining elementary tests corresponding to the elements of the description language.

In [SBC05] the authors present an approach that uses the formal specification of security policies to automatically generate test cases and perform model based testing on the SUT. The tests generated by this method are dependent on the network topology of the tested system and the security requirements specified by the system administrator and they can detect firewalls and configuration errors. However the authors have just presented a theoretical approach and had not described any tool for automating the testing process or to apply on a case study.

3.4.3.2 Passive Testing of Security Policies

The security checking is usually performed using intrusion detection [Den87] systems that employ either misuse detection or anomaly detection. An intrusion detection system is software and/or hardware designed to detect unwanted attempts at accessing, manipulating, and/or disabling of computer systems, mainly through a network, such as the internet. These attempts may take the form of attacks, as examples, by crackers, malware and/or disgruntled employees. An intrusion detection system cannot directly detect attacks within properly encrypted traffic.

An intrusion detection system is used to detect types of malicious behaviours that can compromise the security and trust of a computer system. This includes network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized logins and access to sensitive files, and malwares (viruses, Trojan horses and worms).

An intrusion detection system can be composed of several components [pat]: sensors which generate security events, a console to monitor events and alerts and control the sensors, and a central engine that records events logged by the sensors in a database, and uses a system of rules to generate alerts from security events received. There are several ways to categorize an intrusion detection system depending on the time and the location of the sensors and the methodology used by engine to generate alerts. In many simple intrusion detection system implementations, all three are combined in a simple device or appliance. As an example, SNORT [Roe99] is a free and open source Network Intrusion privation system and network intrusion detection system capable of performing package logging and real time traffic analysis on IP Network. SNORT performs protocol analysis, contact searching/matching, and OS fingerprinting attempts, by dropping attacks as they are taking place. SNORT can be combined with other software such as SnortSnaf, squil , OSSIM, and the basic analysis and security engine (BASE) to provide a visual representation of intrusion data.

In [OC06], a formal security model combining deontic and temporal logic is proposed to specify security properties of OLSR protocols. The model is used to prevent different attacks targeting the link sensing mechanism of the protocol.

Many researchers [ACNn03, GBvS07, BvS01, HGXA06, SL06] are conducted to improve attacks detection for security purposes in various fields. These works are based on different models. Some of them describe the desired behaviour of the system. In this case, the collected traffic that violates this behaviour is considered as a malicious behaviour due to a potential attack. In other methodologies, we only specify the known attacks to avoid any false positive. A false positive arises when an alarm is activated when no attack is performed. Based on this methodology, non-known attacks are not detected.

3.5 Interoperability Testing

Interoperability is defined as the ability of two or more networks, systems, devices, applications or components to exchange information between them and use this information [ETS07]. The purpose of interoperability testing is to give higher confidence on interworking between at least two communicating systems as required by the standards. Although each implementation passes conformance testing, which is to verify that implementations conform to specifications, we cannot guarantee that two implementations can interwork without any problem because they may have different implementation options, some part of specifications can be interpreted in a different way, they may be based on different versions of specifications, etc.

3.5.1 Interoperability Testing Process

In this section, we explain interoperability testing process defined by ETSI [ETS07]. According to the concept of interoperability testing from ETSI, it is assumed that we have an Equipment Under Test (EUT) and a Qualified Equipment (QE) where EUT and QE should come from different suppliers. Interoperability tests are then performed with normal user control and observations, i.e. there is no specialized interfaces for testing purposes and testing is based on functionalities that a user experiments.

One of the important issues is to develop interoperability test cases. As a first step, it is necessary to identify interoperable functions and abstract test architectures. Once interoperable functions and abstract test architectures are identified, we can develop test purposes and define test cases. The steps for developing interoperability test cases are as follows:

1. *Specify an abstract architecture*: this step defines an abstract test architecture where an EUT, QE(s), communication paths between EUT and QE(s), and, if necessary, the expected protocols to be used for communication paths should be clearly identified;

2. *Prepare draft interoperable features statements*: this step identifies whether each function in the standard is mandatory, optional or conditional by other functions;

3. *Specify test suite structure*: this step divides the test suite into test groups based on some logical criteria and defines test coverage within each test group;

4. *Write test purposes*: in this step, a full description of the objective of each test case is specified in its test purpose;

5. *Write test cases*: for each test purpose, detailed test steps that must be followed in order to achieve the test purpose are described. Test cases can be written in either natural languages (e.g. English) or test specification languages (e.g. TTCN-3 (Testing and Test Control Notation version 3)) or programming languages (e.g. C++) or scripting languages (e.g. Perl). If test cases are written in natural language, they should be specified in a clear and unambiguous way. Once we have interoperability

test cases, we can perform the testing process including test planning, test configuration, execution of tests, and producing test reports.

An example of how to practically apply this process can be found in [EMAHC09]. In this work, we conducted an interoperability testing on the functional aspects of the presence service based on the ETSI process. We did not perform any security testing in this step as the object was to practice the interoperability testing on a real case study.

3.5.2 Formal Interoperability Testing

3.5.2.1 Hao *et al*

In the work of hao *et al* [HLSG04], the authors propose a method to generate test cases for interoperability testing of VoIP systems with public switched telephone network.

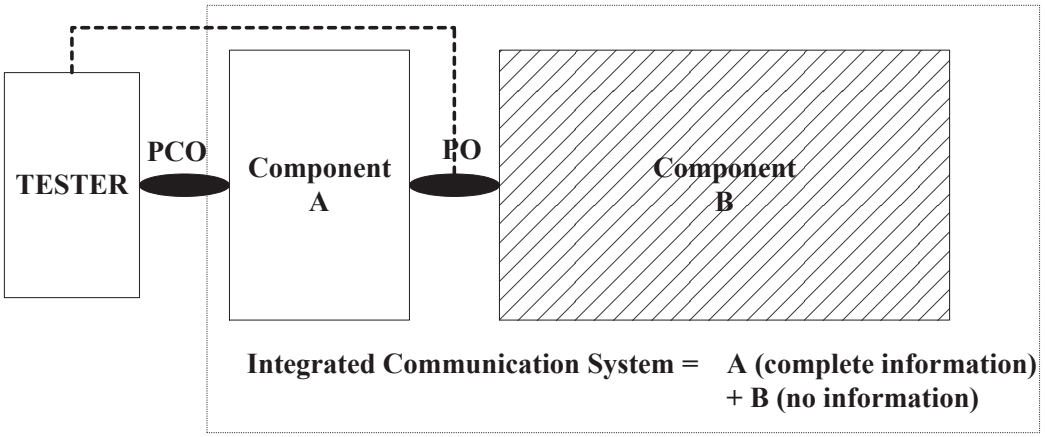


FIGURE 3.5: Interoperability Test Architecture for Embedded Systems.

In their test architecture, the authors consider two components which are illustrated in Figure 3.5: a component A in which the authors have full information, and a component B on which the authors do not or choose not to have any information. For instance the component A can be a pair of phone sets, and component B is the rest of the network. The authors aim at studying the interoperability of component A with B. The authors have represented the systems of component A with Extended Finite State Machines (EFSM). The EFSM that covers the joint behaviour of the several system components can be constructed by calculating the cartesian product of the EFSMs for these components. The test cases are generated from the reachability graph of this model which is a directed graph or a transition diagram. When generating the interoperability tests the authors are concerned only with those failures that occur when the components are interoperating. Thus, the coverage criterion of the interoperability test generation is to test of all the possible interoperations of A with B. Accordingly, the authors define a test case generation algorithm that consists of a test set with a complete coverage and avoid the generation of an infinite

paths which are created due to cyclical paths. The authors defined three steps to generate such a test set:

Step 1) generate all possible acyclic paths, i.e., paths without any repeated vertices.

Step 2) generate all possible simple cycles, i.e., cycles that do not contains any smaller cycles.

Step 3) "Combine" the acyclic paths (from step 1) and simple cycles (from step 2) to generate the final set of paths.

An interoperability test case generation tool is developed based on this algorithm. This tool was used to generate the interoperability test cases for the interoperability testing of traditional plain old telephone service (POTS) feature, authorized phone call feature and the signaling part H.323 of VoIP systems.

Byoun *et al* [BYNc07] have extended this work to generate a test sequence for testing the interoperability, using information contained in the EFSM that specify the communicating system and some graphical properties in the state transition diagram.

3.5.2.2 Desmoulin *et al*

In this work [DV09], the authors have proposed a method for deriving automatically interoperability tests based on formal definitions. According to the authors, interoperability testing has two goals. Firstly, it has to test if the considered implementations communicate correctly. Secondly, it has to verify that during the interaction the implementations provides the expected services. They consider a one-to-one interoperability context which involve only two implementations under test. Also they used the Input-Output Label Transition system to model the specifications of the implementations. In this study, the author propose the interoperability test architecture presented in Figure 3.6.

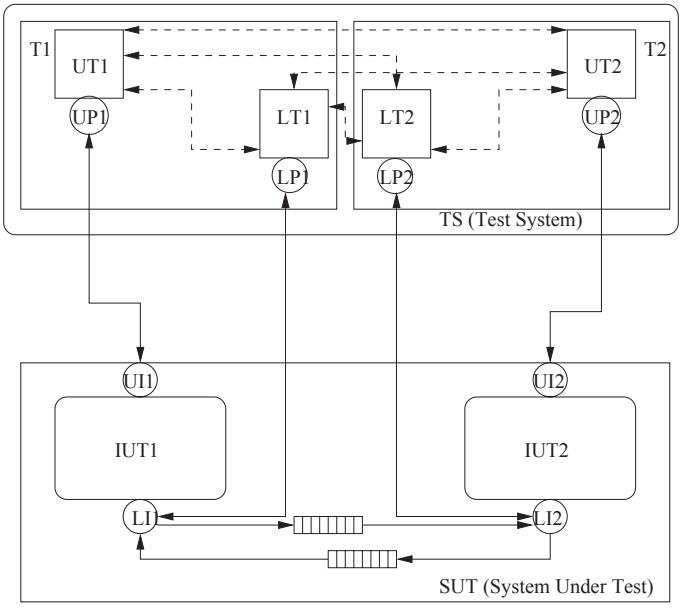


FIGURE 3.6: Interoperability Test Architecture.

In this architecture, UT and LT are respectively an upper and a lower tester. Each of those testers is related respectively to the upper interface (UI) and lower interface (LI) of each implementation under test (IUT).

The authors have propose two formal interoperability definitions called *iop criteria* that give the conditions to be verified by two implementations in order to be considered interoperable.

- The first criterion says that two implementations are considered interoperable if, after a suspensive trace of the asynchronous interaction of the specifications, all outputs and quiescence observed during the interaction of the implementations are foreseen in the specifications.
- The other criterion says that after a suspensive trace of a specification S1 observed during the (asynchronous) interaction of the implementations, all outputs and quiescence observed in the implementation I1 are foreseen in S1, and the same from the point of view of I2 implementing the specification S2.

A new interoperability test generation method is proposed based on these formal definitions. It avoids the well-known state-space explosion problem that occurs when using classical methods.

3.5.2.3 Seol *et al*

When protocol implementations interact with each other, it is possible that additional messages are sent to the implementations, while the previous message is still being processed, and/or simultaneous messages are sent to each implementation at the same time. This is known as "multiple stimuli". The work of Seol *et al* [SKCK04b] considers multiple stimuli in interoperability testing. The communicating entities are modelled using a specific type of FSM knows as input output state machines IOSM. In such IOSMs, a state is said to be one of the following:

- Controllable, if the behaviour of the system can be totally controlled when reaching this state, and this by applying external inputs.
- Uncontrollable, if the behaviour of the system is totally uncontrollable when reaching this state. The behaviour of the system depends on internal inputs.
- Semi-controllable, in this case not all the behaviours of the system can be controlled by applying external inputs.

To cater to the multiple stimuli principle, the authors consider a test architecture with a tester between the two IUTs, Figure 3.7 illustrates the architecture.

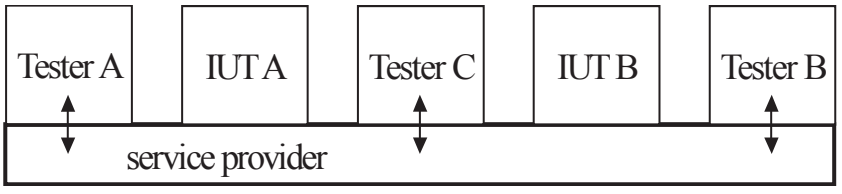


FIGURE 3.7: Interoperability Test Architecture.

The authors have developed an algorithm that generates interoperability test cases with respect to the multiple stimuli. The algorithm starts from the initial stable state consisting of each IOSM's initial states and, at each step, examines all possible external inputs even in the transient states, until a stable state is reached. In the meantime, new paths are generated based on the multiple stimuli principle, and newly found stable states are added to the associated state space. This procedure is repeated for every new stable state until all the stable states are covered. The authors applied their algorithm to the connection establishment phase of TCP and the ATM/B-ISDN signaling protocol.

El-Fakih *el al* [SKCK04a] have also investigated the interoperability testing for multi stimuli models, in their work they developed two methods for the derivation of interoperability test suit. The first method returns a test suite that checks if the implementation is free of livelocks. If the implementation is free of livelocks, the second method returns a test suite that checks if the implementation is conform to the specification.

3.5.3 Security Policy Interoperability Testing

Most of the previous efforts have tried to define models or languages in order to specify interoperability security policy without conflict and redundancy [CCBCB06, CBCBCC08] or to verify some security properties such as consistency, completeness and termination [EKI10].

In the interoperability testing field, all the cited work deal with the functional aspect of communicating systems and cannot be directly applied for interoperability testing of security policies. However, it constitutes the basis for developing a testing approach adapted for the interoperability security policies. New requirements related to the interoperability security policies have to be included in the interoperability testing process. This require the definition of a test architecture adapter for this kind of test. The interoperability security policy has to be described with a formal model that allows automatic generation of test cases. When generating the interoperability test cases, we are interested on specific requirements related to the interoperability security policy. Thus, the test case generation should be guided with a set of test purposes that defines the security objectives to be tested. Finally, when dealing with interoperability testing, in general more then one tester are involved in the test architecture. The existence of multiple testers complicates the test case generation, a set of test cases has to be generated for each tester. Also the execution of the test cases require the synchronization between the different testers which is not an easy task when considering distant systems.

Another possibility, is to inspire form the work done in [MWCM10]. In this work, the authors have used the passive testing technique in order to test the conformance of a security policy with respect to its specification. In this work, some security properties of a communicating systems such as MANET network are tested. The main drawback of this approach is that in some cases the collected trace is not long enough to check some security rules. In this case, we cannot say that the implementation of the security policy conforms with its specification. The test verdict is in this case *inconclusive*.

3.6 Conclusion

At this point we can underline the importance of the role that formal methods can play in order to provide fault free interoperability security policy for communicating systems. This include the following phases: the verification phase showing conceptual faults (livelock, deadlock, etc.), and interoperability testing phase to give higher confidence on interworking between at least two communicating systems as required by the standards.

It has to be noted that conformance testing is outside the scope of this work. However, the work done in conformance testing such as test cases generation and execution can be based on when developing the interoperability testing process for security policies. For instance, one possibility to generate test cases is to construct the model that covers the joint behaviour of the communicating systems. The test cases are generated from the reachability graph of this model which is a directed graph or a transition diagram. As the size of the reachability graph that result from the joint behaviour model is in general large, the use of traditional methods that produces a global reachability graph from which tests are generated is not possible. Therefore, a possibility is to use methods based on partial generation of the reachability graph [CLRZ99] which are basically developed for conformance testing. In these methods test case generation is guided by a predefined set of test objectives. In our case, these test objectives could be the security requirements to be tested. Another possibility is to use the passive testing techniques to test the interoperability security policies. In this case, the communication between the different systems is collected in a trace. This trace is analyzed according to the interoperability security policy specification in order to determine the conformance relation between the implementation and the specification. In this case, the security requirements has to be specified with a formal language.

We will rely in this Thesis on formal models to design a new methodology to make it possible to automatically verify and test interoperability security policies described in a formal language.

Part II

Interoperability Security Policy Integration

"Measure what is measurable, and make measurable what is not so."
Galileo Galilei

Chapter 4

Modeling Contextual Interoperability Security Policies

Contents

4.1 Introduction

Since security policies are considered as critical aspects in modern systems. A security policy is a set of rules that represent the expected behavior of a system in order to maintain a certain level of security. It is necessary to verify whether these systems respect their security policy specifications. This step is mandatory in order to guarantee the global security of a system.

To validate a system against its security policy specification, one has first to model these

policies using formal models. Most of the current works only concentrate on defining a meta language that specify the security policy unambiguously. In Chapter 3 we presented some of these security models and languages. Other existing works have proposed methods to integrate the security policy in the functional model of a system [MOC+07] or to formally model the security policy [HK10, EKI10]. In the second case, in general the functional specification of a system and its security requirements are modeled into two different models and even implemented into two different modules.

The models are used to verify security properties or to check the conformance of the system with respect to its security policy specification. In this Thesis, we rely on several strategies based on formal testing in order to check the conformance relation between the systems and its security policies.

4.2 Overview of the Integration Approach

In this chapter we present our method to integrate interoperability security policies in the functional model of a communicating system [EMHC10]. Our approach is inspired by the work of Mallouli *et al* [MOC+07]. The key idea of this approach is that a security policy restrains the functional behavior of the system in which it is integrated. In some cases the integration of some rules, i.e. obligation rules, adds new behaviors to the system. Such new behaviors have to be properly created and integrated in the model.

Our work distinguishes itself from the existing works in several significant points. First, when dealing with interoperability security policies, it is a hard assumption to consider that the initial model does not contain any security considerations. Therefore our approach considers that the initial model can provide a local security policy. In this case, the integration process must take into account these security considerations, and ensure that this process will be integrated without conflicts with the existing ones. It is also an important assumption when considering dynamic creation and destruction of security rules. In this case, we should be able to revise the integrated security policy each time a security rule is modified. Next, we ensure that our integration process does not create conflict with the functional behavior of the system. Thus, in the case where a security rule is not activated, it should not affect the functional behavior of the system model.

Finally, in our description we consider three different types of security rules modalities: permission, prohibition and obligation, and two different types of contexts: state based and event based context [CCB08]. The model that results from this description will contain both the functional aspect of a system and the security considerations, i.e the interoperability security policies. This model will be used to test if the system respects these interoperability security policies requirements.

4.3 Preliminaries for Security Policies Integration

This section will present the preliminaries needed for our integration process. It specifies the assumptions that we consider and the formal model that we use to specify the functional

model and the interoperability security policies.

4.3.1 Assumptions

In this study, we refer to the *initial model* as the model that does not contain any considerations related the interoperability security policy to be integrated. In particular, this model can contain a local security policy. Later, when this system exchanges information with another one according to an interoperability security policy, a new context is considered. In this new context the initial model is not valid any more since it does not contain the security considerations which are related to the interoperability security policy.

Our approach proposes a method to integrate the interoperability security policy in the functional model of communicating systems. Each system involved in this study is specified using a formal model. In this task we rely on the Extended Finite Automaton (EFA) formalism to model the functional aspect and the security requirements. We assume that the specification of each system is verified with respect to its initial functional requirements. When both systems communicate based on the interoperability security policies, an additional behavior related to the interaction is added to the system models. It is assumed that such behavior exists when integrating the interoperability security policy.

We consider that the interoperability security policies are provided by experts. We also consider that the security policy properties such as consistency, completeness and termination are verified, and are correct. The verification of these security properties is outside the scope of this work.

4.3.2 Extended Finite Automata

In order to model the initial system as well as the security policy, we use the Extended Finite Automata formalism. This formal description is not used only to the control portion of a system but also to properly model the data portion, the variable associated as well as the constraints which affect them. This formalism is detailed and explained in Chapter 3. In the following, we give a reminder of this formalism.

Definition 4 *An Extended Finite Automata (EFA) is an automaton with predicates and action functions. We consider that a transition can be executed only when an input is received and the predicate is true. An extended automaton is a 5-tuple* $E = \langle Q, \vec{v}, T, \Sigma, q_0 \rangle$ *where:*

(i) Q *is a finite set of states and* q_0 *is the initial state;*

(ii) $\vec{v} = (v_1, ..., v_n)$ *is a vector of typed variables;*

(iii) $\Sigma \subseteq (L_i \times L_o)^*$ *is a nonempty set of input/output alphabet where* L_i *and* L_o *are the set of inputs and outputs respectively.*

(iv) T *is a set of transitions. Any transition is a tuple* $\langle q, \sigma, \wp, a, q' \rangle$*, where:*

(i) q *and* q' *are the source and the target states respectively;*

(ii) $\sigma \in \Sigma$ *is the input/output of the transition;*

(iii) a *is an action of the transition. The action updates the values of* $\vec{v}$ *after executing the transition;*

(iv) $\wp : X \rightarrow \{True, False\}$ *is a predicate over the values of* $\vec{v}$*. The transition is activated only if* $\wp$ *is true.*

4.3.3 Characteristic Function

In this study, we use the characteristic function to formally represent the predicate of a transition. Let us consider a set C and a set $A \subseteq C$.

Definition 2 *The characteristic function of* A

$$\chi_A : C \rightarrow \{0, 1\}$$

is defined for all x *in* C *as:*

$$\chi_A(x) = \begin{cases} 1 \text{ if } x \in A \\ 0 \text{ otherwise} \end{cases}$$

The characteristic function of a subset A of a set C, maps elements of C to the range $\{0, 1\}$. In the following we give some useful properties of the characteristic function. Let us denote by A and B two subsets of C, then:

Property 1 $\chi_{A \cap B}(x) = min\{\chi_A(x),\ \chi_B(x)\} = \chi_A(x).\chi_B(x)$
$\chi_{A \cup B}(x) = max\{\chi_A(x),\ \chi_B(x)\} = (\chi_A(x) + \chi_B(x)) - (\chi_A(x).\chi_B(x))$

Let us consider an EFA $\langle Q, \vec{v}, T, \Sigma, q_0 \rangle$ and X the domain of the variable vector $\vec{v}$.

Definition 5 *The domain of the variable vector* $\vec{v}$ *is denoted by* $X = X_1 \times ... \times X_n$*, where* X_i *is the domain of variable* v_i*, is the set of values that a variable can assume.*

Let us consider $W \subseteq X$ a valuation and $t = \langle q, \sigma, \wp, a, q' \rangle$ be a transition where $\wp = true$, then for an $x \in X$, $\chi_W(x) = 1$ and $\wp = false$ when $\chi_W(x) = 0$.

4.3.4 O2O Syntax

In this work we choose to rely on the O2O model to specify the interoperability security policy. This model is detailed in Chapter 2. This section gives a brief reminder on important issues needed in this chapter. We recall that O2O is an extension of OrBAC to manage the interoperability. This choice is motivated by the fact that this model allows specifying security rules without ambiguity. In addition, O2O introduces the concepts of authority sphere which restricts the scope of every security rule to the organization in which the rule

applies, and the management sphere administrating this security policy. Thus, it is possible at any time to know who is in charge of the administration of the interoperability security policy and the scope of this security policy. O2O approach relies on the concept of Virtual Private Organization (VPO) to designate the sub-organization in charge of the interoperability access control. An interoperability security policy defined in a VPO applies to the system that this VPO administrates, namely the O-grantor. Thus, this interoperability security policy has to be integrated in the model that specifies this system. We recall that the system that accesses the O-grantor according to the interoperability security policy of the VPO is called O-grantee. A typical security rule has the following form:

$$SecurityRule\{VPO, modality(role, activity, view, context)\}$$

The *modality* of a security rule can be permission, prohibition or obligation. A *role* is a set of subjects on which the same security rule applies. Similarly, an *activity* and a *view* are respectively a set of actions and objects on which the same security rule applies. The activation of a security rule is dynamic depending on the *context*. Thus, a security rule can be activated only when this *context* is satisfied. We consider two types of context do exist: a state based context and an event based context.

- In the case of a state based context, a security is activated by logical condition provided by the information system like the geographical position of a user.
- In the case of an event based context, the activation of the security rule depends on performing specific actions by users on the information system.

4.3.5 Security Rule Mapping

A permission or a prohibition rule restricts the behavior of the functional model to allow subjects to perform specific actions on objects. Thus, the behavior describing the actions of the security rule exists in the functional model. If the corresponding behavior does not exist in the functional model, it is not necessary to modify the functional model as there is no activity in the model that does not comply with the security policy. The action of an O2O security rule corresponds to a specific request which is represented with an input in the functional model. It is possible that the execution of this input has several possible outputs. In this case, the generated output depends on the input parameters and the system behavior. For instance, if the initial model describes a system that contains a local security policy, an input describing a request to perform an action can be allowed or denied. These two different behaviors are described in two different transitions. Let us consider the following example: *A doctor from a hospital A can read the medical files located in another hospital B*. This security rule has the following notation in the O2O syntax.

$$securityRule\{VPO_{A2B}, permission(doctor, read, medicalFile)\}$$

The action of this security policy could correspond to the transitions described in Figure 4.1.

This Figure shows two transition: the transition t_{SR} describes the permission to read the file, whereas the transition t_{oth} denies this action.

To integrate a security rule we define two set of transitions. Let us consider T_{SR} the set of transitions that correspond to the security rule action. The outputs of those transitions are related to the security rule modality. For instance, if we consider the above example, the transition t_{SR} of Figure 4.1 has to be included in T_{SR}. The other set, T_{oth}, contains those transitions that correspond to the security rule action but they generate outputs which do not comply with the security rule modality.

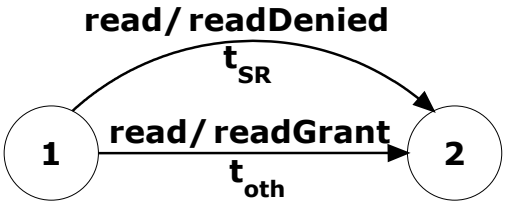


FIGURE 4.1: Example of Mapping the Security Rule Action.

An obligation rule is triggered after the execution of an event in the system. In general, we use obligation rules to specify usage control security. Such security rule could specify a sanction for a user who has performed an unauthorized action. Indeed, the event that activates an obligation rule must exist in the functional model. Otherwise, this security rule should not be integrated as it will be never executed. The action of the obligation rule may not originally exist in this model. In such case, this action is created and integrated in the initial model.

According to the abstraction level of the initial functional model, a subject and an object of a security rule can be represented as global variables or message parameters. A state based context is represented as a condition on some variables of the model. The action of an event based context is related to an input and an output that represent the execution of this action by a user. We denote all the transitions which are triggered by this input and generate this specific outputs by T_e. The subject and object of the event based context can be described as global variables or message parameters.

4.4 Permission Integration

Let us consider a typical permission rule:

$$SecurityRule\{VPO, permission_i(role_i, activity_i, view_i, context)\}$$

This permission rule expresses that a subject (of $role_i$) is allowed to perform an action (of $activity_i$) on an object (of $view_i$) of a system. This security rule is active only when $context$ holds. We recall that in our integration process we consider that the activation of a permission rule may depend on a state based context, denoted $stateContext_i$, or an event based context, denoted $eventContext_i$.

The event based context $eventContext_i$ is described as follows:

$$
\begin{aligned}
&Hold(VPO, s, a, o, eventContext_i) \leftarrow \\
&empower(VPO, S, role_i) \wedge \\
&empower(VPO, A, activity_i) \wedge \\
&empower(VPO, O, view_i) \wedge \\
&After\ do(role'_i, activity'_i, view'_i)
\end{aligned}
$$

This context holds when a subject (of $role'_i$) performs an action (of $activity'_i$) on an object (of $view'_i$) of a system.

Algorithm 1: Permission Integration

Input: - $permission_i(role_i, activity_i, view_i, stateContext_i \wedge eventContext_i)$
- $eventContext_i(role'_i, activity'_i, view'_i)$
- *model*: initial O-grantor model

Output: new O-grantor model

$T_{SR} := searchRule(model, activity_i)$;
foreach *transition* $t_{SR} = \langle q_{SR}, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle \in T_{SR}$ **do**
    $\wp_{SR} := Permit(\wp_{SR}, role_i, view_i, stateContext_i)$;
foreach *transition* $t_{oth} = \langle q_{oth}, \sigma_{oth}, \wp_{oth}, a_{oth}, q'_{oth} \rangle \in T_{oth}$ **do**
    $\wp_{SR} := Prohibit(\wp_{oth}, role_i, view_i, stateContext_i)$;
$T_e := searchContext(model, eventContext_i, T_{SR})$;
// the cases where the behavior of the event based context exists
foreach $t_e = \langle q_e, \sigma_e, \wp_e, a_e, q'_e \rangle \in T_e$ **do**
    $\wp_e := Permit(role'_i, view'_i)$;
    $T'_e := BFS(q_e, 1) \cup BFS(q'_e, 1)$;
    // $BFS(q_e, 1)$ conducts a breath first search from state q_e with depth equal to 1
    $T_{SR} := T_{SR}/T'_e$;
    if $role_i = role'_i$ **then**
        $\wp_{SR} := Constraint(\wp_{SR}, q_e.role_i)$;
// the cases where the behavior of the event based context does not exist
foreach *transition* $t_{SR} = \langle q_{SR}, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle \in T_{SR}$ **do**
    create the transitions t_e and t'_{SR} such that ;
    $t_e := \langle q_{SR}, \sigma_e, \wp_e, a_e, q_e \rangle$;
    $t'_{SR} := \langle q_e, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle$;
    $\wp_{SR} := Permit(\wp_{SR}, role_i, view_i, stateContext_i)$;
    if $role_i = role'_i$ **then**
        $\wp_{SR} := Constraint(\wp_{SR}, q_e.x_s)$;

We divide the integration process into three steps:

- *Step 1:* We integrate the security rule with only the state based context.

- *Step 2:* We integrate the event based context of the security rule.
- *Step 3:* We check whether the event based context and the security rule has to be executed by the same subject. In the case where such relation exists, only the subject that has performed the event based context will be allowed to execute the action of the permission rule.

The integration process is described in Algorithm 1. We consider that v_s, v_o and v_c are the variables that correspond to the subject, the object and the state based context respectively. Thus, X_s, X_o and X_c are respectively the domains of v_s, v_o and v_c.

4.4.1 Step 1

According to the proposed mapping, we can relate the security rule action to two sets of transition, T_{SR} and T_{oth}. The integration process takes into consideration both sets. We will first illustrate the integration of the permission rule in the transitions of T_{SR}. Let us consider a transition $t_{SR} \in T_{SR}$ described as:

$$t_{SR} = \langle q_{SR}, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle$$

The predicate $\wp_{SR}$ is represented by the characteristic function χ_W, where $W = W_1 \times W_2 \times ... \times W_n$ is a set of variable values that gives the predicate $\wp_{SR}$ the value true. Therefore, for an $x \in X$, the predicate $\wp_{SR}$ is satisfied when $\chi_W(x) = 1$. First, we define a new predicate $\wp'$ that has the value true when the conditions related to the permission rule and the functional aspect of the system (not related to the security rule) are satisfied. The predicate can be described by a characteristic function $\chi_{W'}$ where $W' = W'_1 \times W'_2 \times ... \times W'_n \subseteq X$ and defined as:

$$W'_i = \begin{cases} \{role_i\} \text{ if } i = s \\ \{view_i\} \text{ if } i = o \\ \{stateContext_i\} \text{ if } i = c \\ W_i \text{ otherwise} \end{cases}$$

We recall that $role_i$ and $view_i$ describe respectively a set of subjects and objects on which the permission rule is applied. Note that that a permission rule allows to restrain an action and does not relax it. Thus, to integrate the permission rule, we restrain the execution of the transition t_{SR} according to the following scheme. The transition can be triggered when the predicate $\wp_{SR} \vee \wp'$ holds. This new predicate, $new\wp$, of the transition t_{SR} is described as:

$$new\wp : \chi_{W \cup W'}$$
$$t_{SR} = \langle q_{SR}, \sigma_{SR}, new\wp, a_{SR}, q'_{SR} \rangle$$

When the permission rule is active, a transition of T_{oth} should not be executed. Accordingly, we modify the predicate of these transitions to be disabled each time the permission rule is active. A transition $t_{oth} \in T_{oth}$ is described as:

$$t_{oth} = \langle q_{oth}, \sigma_{oth}, \wp_{oth}, a_{oth}, q'_{oth} \rangle$$

The transition t_{oth} should be disabled when $\wp'$ is true. This implies that the transition is enabled when $\neg\wp'$ is true. The predicate $\neg\wp_{oth}$ is described with the characteristic function $\chi_{\overline{W'}}$ where $\overline{W'} = \{x \in X \text{ and } x \notin W'\}$. Thus, the new predicate, $new\wp$, of this transition is a logical conjunction of $\wp_{oth}$ and $\neg\wp'$. This predicate can be described as:

$$new\wp : \chi_W . \chi_{\overline{W'}} = \chi_{W \cap \overline{W'}}$$

This step is described in the Lines 2 to 5 of Algorithm 1

4.4.2 Step 2

For each transition $t_{SR} \in T_{SR}$, where $t_{SR} = \langle q_{SR}, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle$, we search for a transition $t_e = \langle q_e, \sigma_e, \wp_e, a_e, q_{SR} \rangle$ targeting the initial state q_{SR} of t_{SR} and is triggered by an input and generates an output that describes the action of the event based context. If the transition exists, then we modify the predicate of this transition to be allowed for execution for the subjects of $role'_i$ when targeting the objects of $view'_i$ (Lines 7 to 10). The integration process is similar to the first part of step 1. This transition is removed from the set T_{SR}.

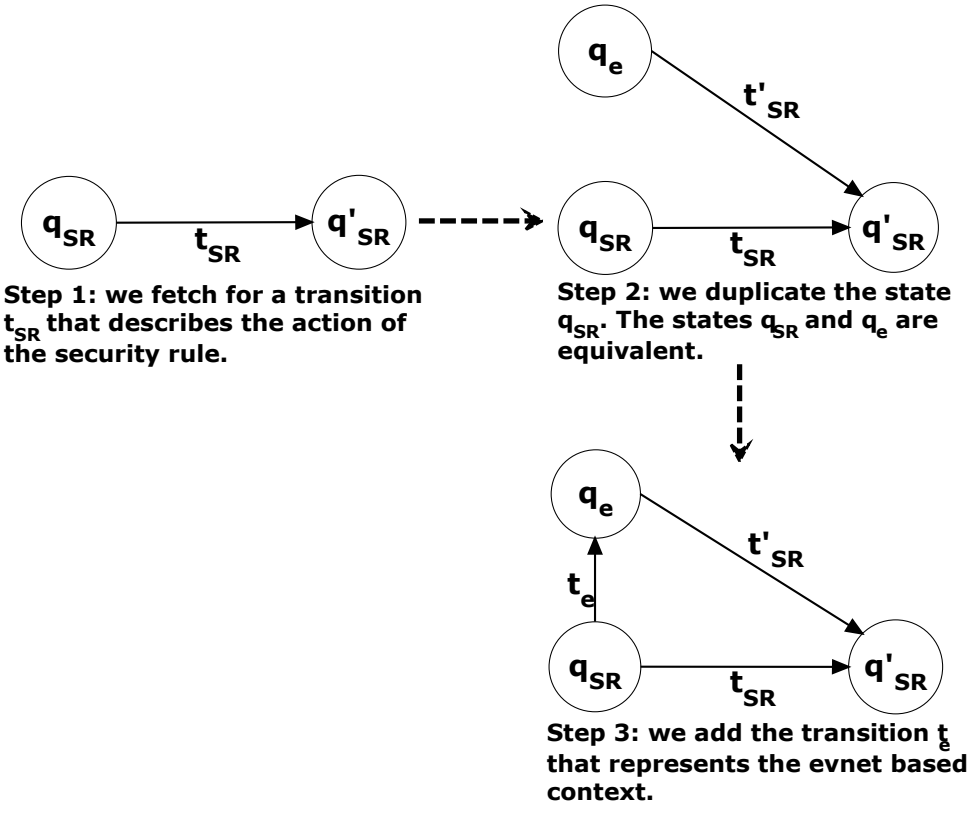


FIGURE 4.2: Event Based Context Integration.

For the rest of the transitions in T_{SR}, the event based context has to be properly integrated in the functional model (Lines 13 to 17). We take into consideration that after

performing the event based context, a user will be able to perform the action described in the permission rule as well as the other actions that can be performed from the state q_{SR}. Also the initial behavior of the system has to be maintained after the integration process. An user who is allowed to execute the transition t_{SR} without performing an event based context (trough the local or the interoperability security policy) has to conserve this privilege after the integration. To integrate the event based context behavior, initially we duplicate the state q_{SR} with another state, denoted by q_e. The states q_{SR} and q_e are now equivalent. The transition $t'_{SR} = \langle q_e, \sigma_{SR}, \wp_{SR}, a_{SR}, q'_{SR} \rangle$ is triggered by the same input, generates the same output, has the same predicate and performs the same actions on the variable vector $\vec{v}$ as the transition t_{SR}. Finally we add the transition $t_e = \langle q_{SR}, \sigma_e, \wp_e, a, q_e \rangle$. The predicate of t_e is defined in such a way to allow the execution of this transition the subjects of $role'_i$ and the objects of $view'_i$. We also modify the predicate of the transition t_{SR}. This transition is disabled for the subjects of $role_i$ and the objects of $view_i$. Figure 4.2 illustrates this step.

4.4.3 Step 3

This step is done after each integration of the event based context such as modification or creation of the transition t_e. Indeed, it is possible, in some cases that the event based context and the security rule is executed by the same user role. In such case an additional constraint must be added to the predicate of transition t_{SR}. Let us denote by x_s the value of variable v_s in state q_{SR} after the execution of the transition t_e. We have that the transition t_{SR} could be executed by an user role if it has has the value x_s. To integrate this condition in the predicate of t_{SR}, we define a new predicate $\wp'$ described with $\chi_{W'}$ where W' is defined as:

$$W_i = \begin{cases} \{x_s\} \text{ if } i = s \\ W_i \text{ otherwise} \end{cases}$$

The final predicate of t_{SR} is the conjunction of $new\wp$ and $\wp'$. This step is represented in the lines 11 to 12 and 18 to 19 of Algorithm 1.

4.5 Prohibition Rule Integration

In the case of a prohibition rule, an event based context specifies that after performing a specific activity, a user will be denied to perform another one on the secured system. The integration of a prohibition rule is similar to the permission rule. In this case, the set T_{SR} contains the transition that describes a prohibition to perform the action of the security rule. Thus, these transitions can be executed when the security rule is active. The set T_{others} contains the other transition mapped by the action of the security rule. That is, users are not allowed to execute the transition when the prohibition rule holds.

4.6 Obligation Rule Integration

In our integration process an obligation rule is triggered by an event based context. The action of the obligation rule is executed after an event is performed on the system. In O2O, an obligation rule has two contexts. A first context that triggers the obligation and thus activates the obligation activity. The other context deactivates the obligation activity. In our case, we assume that obligation rules are directly disactivated after performing the obligation action. In this situation, the second context is always true. Furthermore, we also assume that the behavior of the obligation rules do not exist in the functional model.

Algorithm 2: Obligation Integration

Input: - $obligation_i(role_i, activity_i, view_i, eventContext_i)$
- $model$: initial O-grantor model

Output: new O-grantor model

1 $T_e := searchContext(model, eventContext_i, T_{SR})$;
2 **foreach** $t_e = \langle q_e, \sigma_e, \wp_e, a_e, q'_e \rangle \in T_e$ **do**
3 create the transitions t'_e and t_o such that ;
4 $t'_e := \langle q_e, \sigma_e, \wp_e, a_e, q \rangle$;
5 $t := \langle q, \sigma_o, \wp_o, a_o, q'_e \rangle$;
6 $\wp_e := Prohibit(\wp_o, role'_i, view'_i)$;

Let us consider the following obligation rule:

$$SecurityRule\{VPO, obligation_i(role_i, activity_i, view_i, eventContext_i)\}$$

According to the above mapping, the event based context of the obligation rule maps a set of transitions T_e. Let us note that this set represents the execution of the context action. Let us consider a transition $t_e = \langle q_e, \sigma_e, \wp_e, a_e, q'_e \rangle \in T_e$.

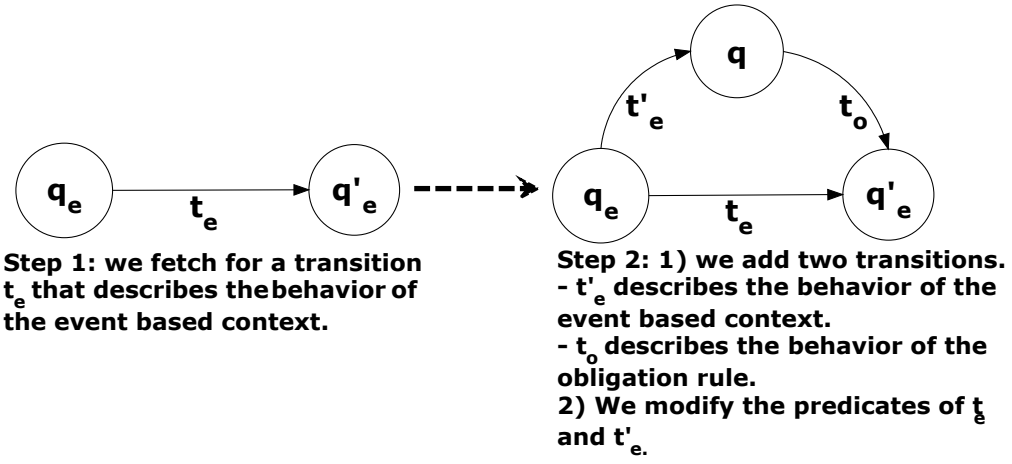


FIGURE 4.3: Obligation Integration

Following, in Figure 4.3 we present an integration of an obligation rule. Accordingly, we create two new transitions. A transition, t_o, that describes the behavior of the obligation

rule, this transition will have q'_e as a target state. The other transition, t'_e is triggered by the same input, generates the same output, has the same predicate and performs the same actions on the variable vector $\vec{v}$ as the transition t_e. However, the target state of t'_e will be the initial state of t_o. Finally, we change the predicate of t'_e to allow its execution only when the conditions on the subject and the object satisfy the event based context specification. The transition t_e should not be executed when the subject and the object verifies the event based context. Algorithm 2 describes the integration process.

4.7 Case Study

We present our approach with a hospital network case study. A hospital network is a set or group of hospitals that work together to coordinate and deliver a broad spectrum of services to their community. We consider the following situation: we have two hospitals, hospital A and hospital B and we assume that each hospital has its local security policy to manage the privileges of the local users. We assume that the following *roles* exist in the two hospitals:

- Doctor: A doctor on duty, he/she can have any speciality.
- Nurse: A nurse on duty, a nurse is in charge of patients within the department he/she belongs.
- Aduser: An user in the administrative staff.
- ITuser: An user in the information technology staff.

Moreover, we also assume that in both hospitals each patient has a medical report, data related for payment and sensitive data which are related to personal information (like previous medical report, insurance company, etc.). Also we assume that each hospital has some system files related to its information system. Moreover, it is possible for a security rule to consider the *roles* as *views*. In such cases the activity of the security rule will target these *roles*. We only consider the interaction from hospital A to hospital B. Thus, the resources to be accessed such as medical reports, sensitive data, payment files or system files might be located in hospital B. Thus, we will index all the roles that comes from hospital A with the letter A. The VPO in charge of the interaction will be denoted by VPO_{A2B}.

4.7.1 Interoperability Security Policy

In this section, we define the local security policy of the two hospitals. In this study, it is assumed that the two hospitals has similar local policies. In the following, we illustrate the local security policy of the hospital B.

4.7.1.1 Local Security Policy of Hospital B

According to O2O the interoperability security policy derives from the local security policy of hospital B. Thus, firstly we specify this local policy. In hospital B we consider the

following activities exist:

- The activity *read* consist of reading a file of hospital B (e.i. medical reports, data related for payment, sensitive data which is related to personal information or system files).
- The activity *edit* consist of modifying a file.
- The activity *add note* consist of adding a note or a comment to a file.

The following context exists in the local security policy:

- *default_ctx*: this context is the default context, it is always true,
- *Fill_PrivacyForm*: this context is true when a nurse fills a privacy form.

The following security rules existes in hospital B:
Rule1: a doctor is permitted to read a medical file.
$SecurityRule\{hospitalB, Permission(doctor, read, MedicalFile, default\_ctx)\}$
Rule2: a doctor is permitted to edit a medical file.
$SecurityRule\{hospitalB, Permission(doctor, edit, MedicalFile, default\_ctx)\}$
Rule3: a doctor is permitted to read a private patient file.
$SecurityRule\{hospitalB, Permission(doctor, read, SensitiveFile, default\_ctx)\}$
Rule4: a doctor is permitted to add notes to the medical files.
$SecurityRule\{hospitalB, Permission(doctor, add\_note, MedicalFile, default\_ctx)\}$
Rule5: a nurse is allowed to read a medical file a medical file after filling a privacy form.
$SecurityRule\{hospitalB, Permission(nurse, read, MedicalFile, Fill\_PrivacyForm)\}$
Rule6: a nurse is allowed to add note the patient file.
$SecurityRule\{hospitalB, Permission(nurse, add\_note, MedicalFile, default\_ctx)\}$
Rule7: an AdUser is permitted to read a payment file.
$SecurityRule\{hospitalB, Permission(AdUser, read, PaymentFile, default\_ctx)\}$
Rule8: an AdUser is allowed to edit a payment file.
$SecurityRule\{hospitalB, Permission(AdUser, edit, PaymentFile, default\_ctx)\}$
Rule9: an AdUser is allowed to add note to a payment file.
$SecurityRule\{hospitalB, Permission(AdUser, add\_note, PaymentFile, default\_ctx)\}$
Rule10: an ITUser is permitted to read a system file.
$SecurityRule\{hospitalB, Permission(ITUser, read, SystemFile, default\_ctx)\}$
Rule11: an ITUser is allowed to edit a system file.
$SecurityRule\{hospitalB, Permission(ITUser, edit, SystemFile, default\_ctx)\}$
Rule12: After editing a medical report by a doctor from Hospital A, the doctor that is in charge of this patient will be notified.
$SecurityRule\{hospital, Obligation(System, notify, DoctorInCharge, modify\_report)\}$

The context $Fill\_PrivacyForm$ is defined as follows:
$Hold(hospitalB, S, A, O, Fill\_PrivacyForm) \leftarrow$
$empower(hospitalB, S, nurse) \wedge$
$empower(hospitalB, A, read) \wedge$
$empower(hospitalB, O, MedicalFile) \wedge$
$After\ do(S, fill, PrivacyForm)$

4.7.1.2 Interoperability security policy

In our case study, we aim to test the interoperability security policy of hospital B. Thus, we consider only the interaction from hospital A to hospital B. Therefore, only the virtual private organization of hospital B is needed, denoted VPO_{A2B}. This VPO has two attributes: the O-grantee (hospital A) and the O-grantor (hospital B). We consider that hospital B does not define any restrictions on the local resources when it operate with hospital A, this is formally defined as:
$use(VPO_{A2B}, O, V) \leftarrow use(S1, O, V)$

The following definition of roles exists in the VPO_{A2B} :
$empower(VPO_{A2B}, S, doctor_A) \leftarrow empower(hospitalA, S, doctor)$
$empower(VPO_{A2B}, S, nurse_A) \leftarrow empower(hospitalA, S, nurse)$
$empower(VPO_{A2B}, S, ITUser_A) \leftarrow empower(hospitalA, S, ITUser)$
$empower(VPO_{A2B}, S, AdUser_A) \leftarrow empower(hospitalA, S, AdUser)$

The following security rules are defined in the VPO_{A2B}:
Rule13: a doctor of hospital A is permitted to read a medical file.
$SecurityRule\{VPO_{A2B}, Permission(doctor_A, read, MedicalFile, default\_ctx)\}$
Rule14: a doctor of hospital A is permitted to read a private patient file after signing a non divulgation form.
$SecurityRule\{VPO_{A2B}, Permission(doctor_A, read, SensitiveFile, SignNDF)\}$
Rule15: a doctor of hospital A is permitted to edit a medical file of his patients.
$SecurityRule\{VPO_{A2B}, Permission(doctor_A, edit, MedicalFile, default\_ctx)\}$
Rule16: a doctor of hospital A is permitted to add notes to the medical files.
$SecurityRule\{VPO_{A2B}, Permission(doctor_A, add\_note, MedicalFile, default\_ctx)\}$
Rule17: a nurse of hospital A is allowed to read a medical file a medical file after filling a privacy form.
$SecurityRule\{VPO_{A2B}, Permission(nurse_A, read, MedicalFile, Fill\_PrivacyForm)\}$
Rule18: a nurse of hospital A is allowed to add note the patient file.
$SecurityRule\{VPO_{A2B}, Permission(nurse_A, add\_note, MedicalFile, default\_ctx)\}$
Rule19: an AdUser of hospital A is permitted to read a payment file.
$SecurityRule\{VPO_{A2B}, Permission(AdUser_A, read, PaymentFile, default\_ctx)\}$
Rule20: an AdUser of hospital A is allowed to edit a payment file.
$SecurityRule\{VPO_{A2B}, Permission(AdUser_A, edit, PaymentFile, default\_ctx)\}$
Rule21: an AdUser of hospital A is allowed to add note to a payment file.
$SecurityRule\{VPO_{A2B}, Permission(AdUser_A, add\_note, PaymentFile, default\_ctx)\}$
Rule22: an ITUser of hospital A is permitted to read a system file.
$SecurityRule\{VPO_{A2B}, Permission(ITUser_A, read, SystemFile, SystemError)\}$
Rule23: an ITUser of hospital A is allowed to edit a system file.
$SecurityRule\{VPO_{A2B}, Permission(ITUser_A, edit, SystemFile, SystemError)\}$
Rule24: After editing a medical report by a doctor from hospital A, the doctor that is in charge of this patient will be notified.
$SecurityRule\{VPO_{A2B}, Obligation(System, notify, DoctorInCharge, modify\_report)\}$

The contexts $SignNDF$, $Fill\_PrivacyForm$ and $SystemError$ are defined respectively as follows:
$Hold(VPO_{A2B}, S, A, O, SignNDF) \leftarrow$
$empower(VPO_{A2B}, S, doctor_A) \wedge$
$empower(VPO_{A2B}, A, read) \wedge$
$empower(VPO_{A2B}, O, SensitiveFile) \wedge$
$After\ do(S, sign, NDF)$

$Hold(VPO_{A2B}, S, A, O, Fill\_PrivacyForm) \leftarrow$
$empower(VPO_{A2B}, S, nurse_A) \wedge$
$empower(VPO_{A2B}, A, read) \wedge$
$empower(VPO_{A2B}, O, MedicalFile) \wedge$
$After\ do(S, fill, PrivacyForm)$

$Hold(VPO_{A2B}, S, A, O, SystemError) \leftarrow$
$empower(VPO_{A2B}, S, ITUser) \wedge$
$empower(VPO_{A2B}, O, SystemFile) \wedge$
$SystemError()$

4.7.2 Formal Model

The hospital A and hospital B are modeled using the IF language [BGM02]. A communicating system described using IF language is composed of active process instances running in parallel and interacting asynchronously through shared variables or signals which can be send via signal routes or by direct addressing. A process instance can be created and destroyed dynamically during the system execution. Each process has local data and a private FIFO buffer. Each IF process is described as a finite automaton extended with variables. Using the IF language we specified a partial model of hospital B. This partial model, denoted as *initial model*, does not specify all the behavior of the hospital but a partial one which is involved by the interoperability security policy that we would like to integrate. Note that the initial model already contains a local policy. This security policy can also be integrated according to our method. In the initial models, each of the two hospitals contains five subjects for each role, and ten objects for each view. hospital B model has three tables that specify the doctor in charge of each medical report, the nurse that is responsible of each medical report and the sensitive information that are related to each medical report.

4.7.3 Interoperability Security Policy Integration

In this section we present how we integrate the O2O rules in the initial model of hospital B. Figure 4.4 represents the hospital B model in which the interoperability security policy is integrated. The behavior of the resulted model is different from the initial model. The behavior of this model is now constrained by the interoperability security policy.

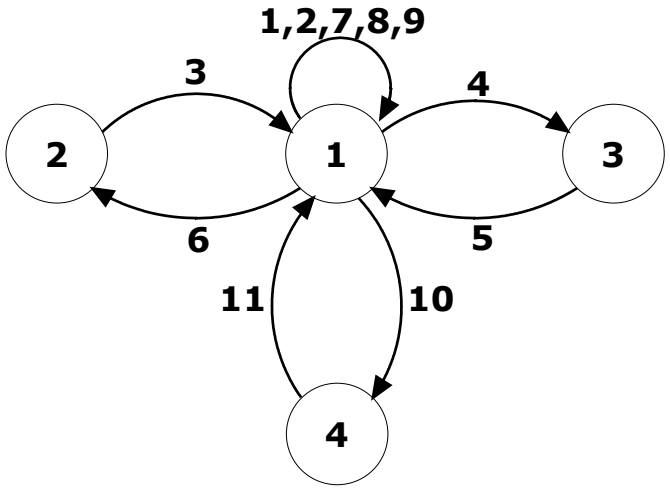


FIGURE 4.4: Initial Model of Hospital B.

1	?access_file(s,o)/!access_grant() if(((s ∈ doctorA ∨ s ∈ doctor)∧ o ∈ MedFile) ∨ (s ∈ ITuserA ∧ o ∈ SysFile ∧ SystemError) ∨ (s ∈ ITuser ∧ o ∈ SysFile) ∨ ((s ∈ AdUserA ∨ s ∈ AdUser) ∧ o ∈ PaymentFile)))∧ (s ∉ doctorA ∨ o ∉ RestrictedInf)∧ (s ∉ nurse ∨ o ∉ MedFile)∧ (s ∉ nurseA ∨ o ∉ MedFile)
2	?add_note(s,o)/!add_your_note() if(((s ∈ doctorA ∨ s ∈ doctor ∨ s ∈ nurse ∨ s ∈ nurseA) ∧ o ∈ MedFile) ∨ ((s ∈ AduserA ∨ s ∈ AdUser) ∧ o ∈ PaymentFile))
3	?sign_NDF(s,o)/!NDF_signed() if((inCharge(s,o) ∨ s ∈ doctorA ∧ s ∈ doctor) ∧ o ∈ RestrictedInf)
4	?edit_file(s,o)/!edit_grant() if((s ∈ doctorA ∨ s ∈ doctor) ∧ o ∈ MedFile)∧ (s ∉ nurseA ∨ o ∉ MedFile)
5	?notify_inCharge(s,o)/!notified()
6	?access_file(s,o)/!access_grant() if(s ∈ doctorA ∧ o ∈ RestrictedInf)
7	?access_file(s,o)/!denied() if(((s ∈ ITUser ∨ s ∈ doctor ∨ s ∈ nurse)∧ o ∈ PaymentFile) ∨ ((s ∈ ITUser ∨ s ∈ AdUser ∨ s ∈ nurse)∧ o ∈ MedFile)∨ ((s ∈ doctor ∨ s ∈ AdUser ∨ s ∈ nurse)∧ o ∈ SystemFile))∧ (s ∉ doctorA ∨ o ∈ MedFile) ∧ (s ∉ ITuserA ∨ o ∉ SysFile ∨ ¬SystemError) ∧ (s ∉ AdUserA ∨ o ∉ PaymentFile))

8	?add_note(s,o)/!denied()
	if((s ∈ nurse ∨ s ∈ doctor) ∧ o ∈ PaymentFile)∨
	if((s ∈ ITUser ∨ s ∈ AdUser) ∧ o ∈ MedFile)

9	?edit_file(s,o)/!denied() if(((s ∈ nurseA ∨ s ∈ nurse) ∧ o ∈ MedFile))∧ (s ∉ doctorA ∨ o ∉ Medfile)
10	?fill_privacyForm(s,o)/!PrivacyForm_filled() if(s ∈ nurseA ∧ o ∈ MedFile)
11	?access_file(s,o) /!access_grant() if(s ∈ nurseA ∧ o ∈ MedFile)

In Table 4.1 we present some metrics of the integration process.

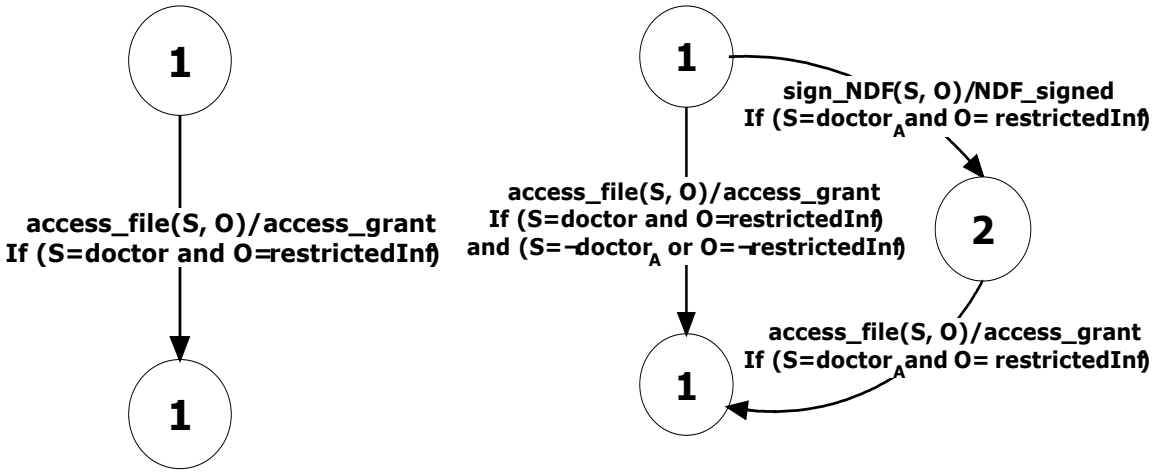
	# states	# transitions	# signals
Before integration	3	7	9
After integration	4	11	13

TABLE 4.1: Integration Results

In the following we illustrate the integration results of a permission, a prohibition and an obligation rules. It is important to mention that these example are defined in our interoperability security policy, and presented in the model that is depicted in Figure 4.4.

4.7.3.1 Permission Rule Integration

The first example deals with a permission rule: *a doctor from hospital A may have access to restricted and more sensitive information of his/her patient after filling a non divulgation form.* In our initial model, a doctor of hospital B has direct access to the restricted information. Thus, no action is required before he/she accesses this information. This is represented in the *initial model* as a transition that starts at the state 1 and targets the same state. This transition is presented in Figure 4.5a. The integration process follows the following steps. In the initial state of the transition describing the security rule activity, there are added two new messages *sign_NDF* and *NDF_signed* as new signals (with parameters and signal routes). If actions on some variables of the model are needed, these actions are also added in this step. After the deployment of the integration algorithm on the old specification we obtain a new one described in Figure 4.5b.



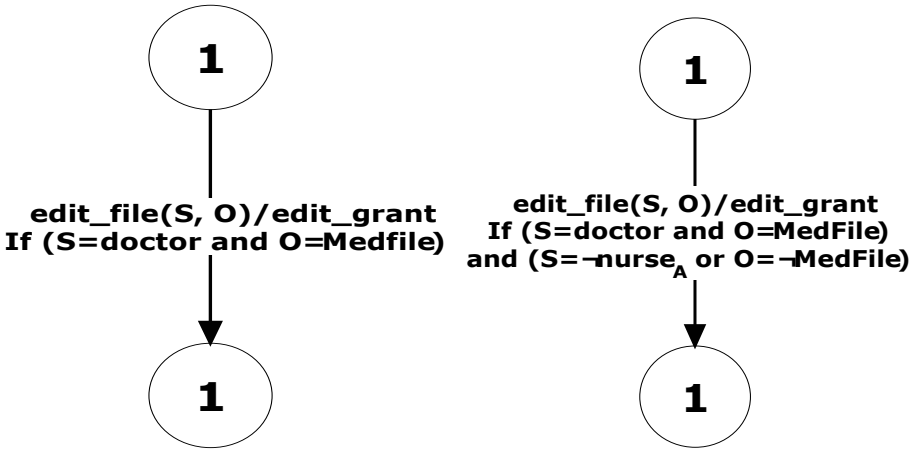
(A) Initial Transition

(B) Integration of the Permission Rule.

FIGURE 4.5: Permission Rule Integration.

4.7.3.2 Prohibition Rule Integration

A prohibition rule is integrated in the same way as a permission rule. In this case the predicate of the transition that describes the security rule activity is restrained in order to disable this transition when the security rule is satisfied. According to the interoperability security policy: *a nurse from hospital A is not allowed to edit a medical report.*



(A) Initial Transition

(B) Integration of the Prohibition Rule.

FIGURE 4.6: Prohibition Rule Integration.

Figure 4.6a shows the transition describing the edit activity. Note that in our *initial model*, this transition starts at the state 1 and targets the same state.The predicate of this transition has to be modified to restrict a nurse of hospital A from editing a medical report. Figure 4.6b presents the integration of a prohibition rule in the initial transition of the functional model.

4.8 Conclusion

In this chapter we presented a method to integrate a contextual based interoperability security policy in the functional model of a system. In our approach we consider that the initial functional model can be used to describe a system and its local security policy. Our method ensures that the integration process does not create conflicts with the local policy and the functional aspects of the initial model. Where we have the integration, the obtained model describes a system that respects the interoperability security policy considerations. This new model can be used to apply formal methods such as test case generation and interoperability testing to verify the conformance of this system with respect to the interoperability security policies. In particular, our integration method allows the specification of different modalities such as obligation, permission, and prohibition and two types of context state based and event based context. Moreover, by using our method, an integrated security policy can be easily updated and revised. It can be done by only integrating the new requirements in the system model. In the next chapter we propose an

approach to verify the correctness of a security policy which is integrated in the functional model of a system.

Part III

Security Policy Verification

"An error does not become a mistake until you refuse to correct it."
Orlando A. Battista

Chapter 5

Formal Verification of Interoperability Security Policies

Contents

5.1 Introduction

After describing a system with a formal model, it is strongly necessary to verify whether the behavior of this model respects its security policies requirements. Without this step, we cannot rely on this model to perform formal testing of security policies as there is no guarantee if this model respects these requirements.

Most of the previous work in this domain focus on the verification of security properties such as consistency, completeness and termination [HK10]. In these studies, the security policy is modeled without the functional aspect of the system on which it applies. In other work, these models are used to check the correctness of this model with respect to the security properties [EKI10]. However, when we consider dynamic security polices, the activation of each security rule can be constrained with contexts. In general, a security rule context involves the functional aspect of a system. Thus, modeling only the security

policy is not sufficient anymore. This model should include both the functional aspect and the security policy. In this case, the existing methods cannot be applied anymore as it should be able to verify whether the system model respect the security policy according to a context. For instance, if a hospital grants access to its patient files for doctors from another hospital in the context *interoperability*, this rule is not activated if this context does not hold. Therefore, doctors have access to the patient file only in the context *interoperability*.

5.2 The Key Idea

In this chapter we propose an approach to verify the correctness of a system model with respect to its interoperability security policies using model checking techniques [EMCHZ11]. We consider OrBAC security policies and we base on the O2O model for managing the security interoperability. According to the O2O model, an interoperability security policy defines the rights of the users of an organization (the O-grantee) when accessing the resources of another organization (the O-grantor). let us recall that the scope of this interoperability security policy is limited to the organization on which it applies, which is the O-grantor organization. Therefore, in this study only the functional model of an O-grantor system is needed. The model on which we apply the formal verification describes the functional aspect and the interoperability security policies of the O-grantor.

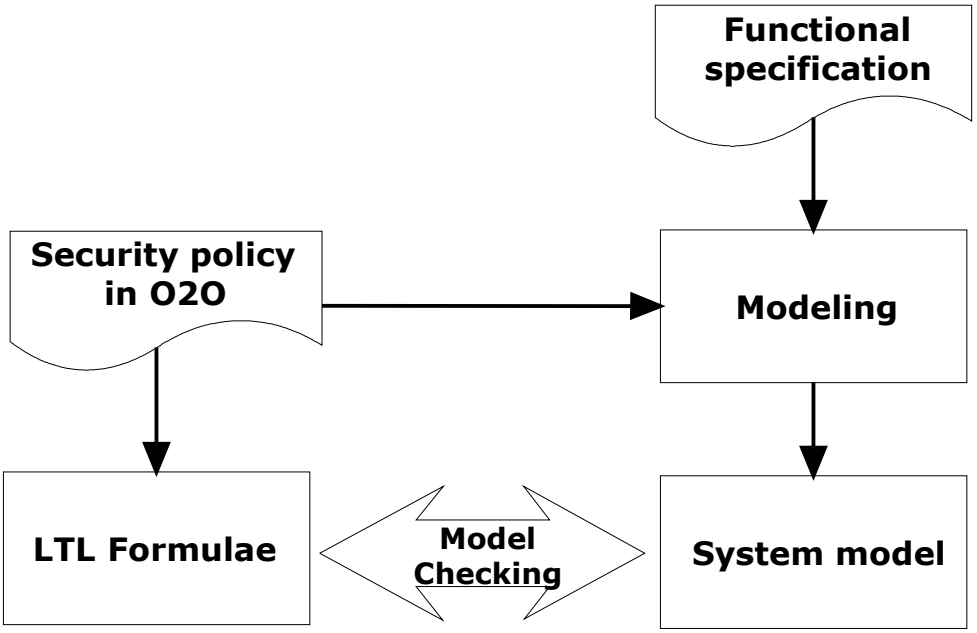


FIGURE 5.1: Verification Framework.

The OrBAC language cannot be directly used to verify the correctness of the security policy. Thus, we propose to transform the security rules specified based on the O2O model into the Linear Temporal Logic (LTL). This logic allows to formalize properties that can be checked in a system execution using a model checker such SPIN [Hol03]. In particular we will verify that all the executions of the model in which the policy is integrated respect

these properties. The choice of the SPIN model-checker is motivated by the fact that it is an efficient and well accepted model-checker that relies on well-established algorithms that can manage huge specifications. It provides several means to handle the state space explosion such as compression algorithms, different levels of abstraction by states fusion, a bit-state space vector and so on. Figure 5.1 illustrates the proposed framework.

5.3 Describing O2O Security Rules with LTL

5.3.1 Assumptions

We assume that the activation of a security rule can be contained with a state based context or an event based context. The former context describes logical conditions, whereas the latter describes an action that has to be performed in order to activate the security rule. Note that when modeling the security policy with the functional behavior of a system, the differentiation between these two types of contexts depends on the level of abstraction of the model. The condition of an event based context expresses a behavior in a system model that has to be performed for activating the security rule. A state based context is represented in the system model with boolean variables or functions given by an oracle whether if some conditions are satisfied or not. In this Thesis it is assumed that the system model and the security rules are specified in the same level of abstraction.

Finally, in this study we assume that the interoperability security policy is consistent and complete. Such security properties are considered outside the scope of our study. A possible way for designing consistent security policies is by using the MotOrBAC tool [CCBC06].

5.3.2 Linear Temporal Logic

In this section we will introduce the linear temporal logic (LTL) [Hol03]. The logic allows to formalize properties of a path in a computation tree unambiguously and concisely with the help of a small number of special logical operators and temporal operators. Given a set of atomic propositions $\wp$ where $\{\phi, \varphi\} \subseteq \wp$, LTL formulae are constructed inductively as follows. If ϕ and φ are formulae then:
Logical operators:

1. $\phi \wedge \varphi$ (logical conjunction), $\phi \vee \varphi$ (logical disjunction) are also formulae and $\neg\phi$ (logical negation) is a formula.
2. $(\phi \longrightarrow \varphi)$ (logical implication) and $(\phi \longleftrightarrow \varphi)$ (logical equivalence) are formulae.
3. $\top$ and $\bot$ denotes true and false respectively.

Temporal operators:

1. $\circ\phi, \Diamond\phi, \Box\phi, \phi R\varphi$ and $\phi \cup \varphi$ are formulae.

where
○(next) the formula $\circ\phi$ holds if the formula ϕ holds in the next state.
□ (always) the formula $\Box\phi$ holds if the formula ϕ holds in all the states of the path.
♢ (eventually) the formula $\Diamond\phi$ holds if the formula ϕ eventually holds in a state of the path.
R (release) the formula $\phi R\varphi$ holds if either φ holds globally in the path or ϕ occurs before the first state at which φ is violated.
∪ (until) the formula $\phi \cup \varphi$ holds, if ϕ holds until φ occurs, i.e., there is a state on the path at which φ holds, and at every state before ϕ holds.

5.3.3 Decomposition of an O2O Security Rule

An O2O security rule can be decomposed into the following three entities:

- An active security rule: the security rule is considered active, which means that this rule is not constrained by a context or the context is satisfied.
- A state based context.
- An event based context.

In this section we define the LTL formulae that describes each of the above entities. The motivation behind this decomposition is to show how each part of a contextual based security policy can be represented by using LTL formulae. The results of this description will be used to compose a contextual based security rules in LTL. Note that this decomposition is useful when not all the security rules are constrained with a state based or an event based context.

5.3.3.1 Describing an Active Security Rule

An O2O security rule is defined as follows:

$$securityRule(VPO, modality(role, activity, view, context))$$

In this step it is considered that the *context* of the security rule is always true. We denote by *Subject* and *Object* the formulae that represent respectively a subject of *role* and an object of *view*. We denote by Req_r, the formula that represents the request to perform an action of *activity*. This request can generate several possible outputs in the functional model. For instance this request can be granted or denied. We denote by Ack_{Req_r} the formula that describes the expected output generated after the execution of the security rule. The other possible outputs that can be generated from the request are represented by the formula $NAck_{Req_r}$. In the case where the *modality* of the O2O security rule is a permission or a prohibition, the security rule is active when the value of the request, subject and object meets this rule. The security system generates a specific answer when the security rule is executed. Formally, this can be expressed with an LTL formula as

follows:

$$Permission\ or\ prohibition : Req_r \wedge Subject \wedge Object \rightarrow \Diamond(\neg NAck_{Req_r} \cup Ack_{Req_r})))$$

An obligation rule represents an action that has to be performed by a subject on an object of the secured system. Thus, it is formally defined using LTL as follows:

$$Obligation : (Req_r \wedge Subject \wedge Object) \wedge ((Req_r \wedge Subject \wedge Object) \rightarrow \Diamond(\neg NAck_{Req_r} \cup Ack_{Req_r})))$$

5.3.3.2 Describing a State Based Context

A state based context is defined in O2O as follows:

$$Hold(VPO, subject, action, object, context_s) \leftarrow conditions$$

This description means that $context_s$ is true for a specific subject, object and action and when the logical *conditions* are true. Let us denote by C_s the formula representing the logical *conditions* of the state based context. Thus, this context can be described in LTL as follows:

$$Context_s : \ C_s \wedge (C_s \rightarrow Req_r \wedge Subject \wedge Object)$$

Thus if the logical conditions specified in C_s does not occur in a system execution, $context_s$ remains false. In the case where the formula C_s is satisfied, the context is only true for a specific subject object and action which are respectively specified in the formulae $Subject$, $Object$ and Req_r.

5.3.3.3 Describing an Event based context

In the case of a permission rule, an event based context expresses a pre-obligation which specifies an action that has to be performed in order to activate the permission rule. In the case of a prohibition rule, an event based context specifies that after performing a specific activity, an user will be prohibited to perform another one on the secured system. An obligation rule is triggered when specific action is executed on the secured system. O2O defines an event based context as follows:

$$Hold(VPO, subject, action, object, context_e) \leftarrow After\ do(role', activity', view')$$

Thus, in this case the context $context_e$ is true for *subject*, *object* and *action* in the state which follows the occurrence of the action $do(role', activity', view')$. We denote by C_e the

formula that describes this action. The event based context is represented in LTL as:

$$context_e : \ C_e \wedge (C_e \rightarrow Req_r \wedge Subject \wedge Object)$$

In particular, if the action described in C_e does not occur in a system execution, then $context_e$ remains false. In the case where the formula C_e is satisfied, then the context is holds for a specific subject, object and action which are verified in the formulae $Subject$, $Object$ and Req_r respectively. Let us denote by $Subject'$, $Object'$ and Req_e the formulae representing respectively a subject of $role'$, an object of $view'$ and an action of $activity'$. The formula Ack_{Req_e} represents the expected output specifying that the action of the event based context is successfully performed. The formula $NAck_{Req_e}$ describes the other possible output that can be generated by the secured model in response to this request. The formula C_e is satisfied only when the behavior that it describes is performed. So, we can define C_e with the following LTL formula:

$$(Req_e \wedge Subject' \wedge Object') \wedge ((Req_e \wedge Subject' \wedge Object') \rightarrow \\ \diamondsuit(\neg NAck_{Req_e} \ \cup \ Ack_{Req_e}))$$

5.3.4 Contextual Based Security Rule Transformation

In this section we define the LTL formulae describing the contextual based interoperability access control security rules. The notation of an O2O security rule is given in the following example:

$$securityRule(VPO, modality(role, activity, view, context_s \wedge context_e))$$

We aim to verify the correctness of contextual based security rules. So, the LTL formulae that describe a security rule have to satisfy the following:

- If in a system execution the request (described by $Req_r \wedge Subject \wedge Object$) that activates a security rule is not satisfied, then the LTL formulae describing this security rule have to be satisfied as in this case the correctness of the security rule could not be violated.
- If the event based context and the state based context are satisfied, then the security rule should be activated. In particular, we have that when the security rule is executed, a specific response should be generated by the system.
- If the state based context or the event based context are not satisfied, then the security rule should not be active. That is, this rule cannot be executed.

In the case of a non active permission or prohibition rule, we have that it cannot be executed. Thus, the output described in Ack_{Req_r} should not be generated in response to $Req_r \wedge Subject \wedge Object$. We describe this behavior in LTL formalism as follows:

$$not\_active\_permission \ or \ not\_active\_prohibition : \\ Req_r \wedge Subject \wedge Object \rightarrow \diamondsuit\neg Ack_{Req_r}$$

Next we present the behavior described in an obligation rule. The conditions to triger the transition can be defined in a state based context or event based context. The LTL formula that describes a non active obligation rule is defined as follows:

$$not\_active\_obligation: \ \neg \ Req_r \wedge Subject \wedge Object$$

A contextual based permission security rule is described with two LTL formulae:

$$F1: \ context_e \rightarrow (context_s \rightarrow permission)$$
$$F2: \ (\neg context_e \vee \neg context_s) \rightarrow not\_active\_permission$$

On one hand, the formula $F1$ describes that a permission security rule can be executed only if the two contexts are satisfied. In the other had, in the case where one or both of the contexts are not satisfied, the formula F2 verifies that the permission rule is not active. Moreover, if one or both of the context types are not required for a permission rules, we simply replace the formulae $context_e$ and/or $context_s$ with the formula $\top$. The formulae defining a contextual based prohibition or obligation rule are similar to the permission rule case.

5.3.4.1 LTL Formulae Instantiation

In this section we discuss how to instantiate the LTL formulae that are describing a set of O2O security rules. This approach is applied in two steps. First, we relate the O2O parameters to elements that correspond to the functional model. Then, we use the previous computed values to instantiate the LTL formulae.

An action of an O2O security rule can be related to a specific request and the possible outputs that can be generated by this request. Then, it is possible to narrow down the number of outputs to one specific output by considering the security rule modality. The subject and object of the security rule are represented as variables in the functional model or parameters in the request message. A state based context is represented as a condition on some variables in the functional model. It provide an oracle whether the context is satisfied or not. An event based context represents a behavior that has to be performed in order to activate the security rule. Thus, an action of the context is related to a request and an output describing that the action is successfully performed. A subject or an object of the context are described with variables or messages parameters in the functional model. This mapping can be done by using a table that relates each element of an O2O security rule to the element that corresponds in the functional model.

Following, we use the mapping result to instantiate a LTL formula. For instance, the formula $Req_r \wedge S \wedge O$ represents a request sent from a specific subject targeting a specific object. Thus, using the mapped values of the action, the subject and the object of an O2O security rule, it is easy to define the LTL formula that verifies this property.

5.4 Security Policy Verification

In this section we present our verification process through a hospital network case study. In the previous chapter we show how to integrate interoperability security policies in the functional model of a system. We applied our integration process to a hospital network case study. In this section, we will use the generated model after the integration process in order to verify whether the models respect the interoperability security policy.

5.4.1 Defining the Verification Process

We aim to verify the correctness of the interoperability security policy which is integrated in the O-grantor model against its specification modeled using O2O. Thus, the system model whose executions has to satisfy the access control security policy objectives consists of the O-grantor system which contains the interoperability security policy. The other entities that interact with this model, such as the local entities the local entities of this model and the O-grantee model are "the environment".

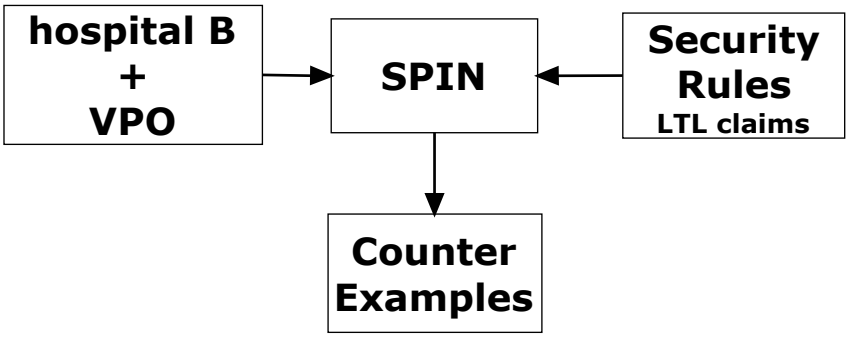


FIGURE 5.2: Verification Process.

5.4.2 Application to the Hospital Network Case Study

Next we continue extending our approach based on hospitals networks presented in Chapter 4. We show in the previous chapter how the interoperability security policy can be integrated in the functional model of Hospital B. We will use this model to formally verify the behavior of this model respects its interoperability security policy.

In this case study Hospital B model represents the O-grantor. Thus, it represents the model which executions have to verify the interoperability security policy. Figure 5.2 illustrates the verification process.

5.4.3 Description of the Interoperability Policy Using LTL

In our case study, we defined 13 interoperability security rule. We generated for each security rule two LTL formulae (Table 5.1 summarizes the results).

Interoperability scurity policy	O2O	LTL
Set of requirements	13 security rules	26 formulae

TABLE 5.1: Results of the Transformation of the Interoperability Security Policy.

The LTL formulae are then instantiated with the values of subjects and objects that activate the original O2O security rule. Consider the requirement "*The system is obligated to notify a doctor after editing one of his/her patient's medical reports by a doctor from hospital A*". It specifies an obligation rule with an event based context. The O2O notation of this rule is given in the following:

$$securityRule(VPO_{A2B}, obligation(system, notify, doctor, MedReportEdit))$$

The context $MedReportEdit$ can be described as:

$$hold(VPO_{A2B}, system, notify\_Edit, O, MedReportEdit) \leftarrow$$
$$After\ do(S', edit\_file, x.pdf)$$

In our model each role and view are represented with a sequence of values which will be assigned to subjects and object of these roles and views respectively. We also specify the doctor in charge of each medical report. Table 5.2 gives the mapping that corresponds to this example. We defined the following formulae:

$obligation$: $fromEnv?[notify\_inCharge(s,o)]\&\&$
$(fromEnv?[notify\_inCharge(s,o] \rightarrow \Diamond$
$toEnv?[notified(0,0)])$

C_e : $fromEnv?[edit\_file(s',o')]\&\&(fromEnv?[edit\_file(s',o')]$
$\rightarrow \Diamond toEnv?[edit\_grant(0,0)])$

$context_e$: $C_e\&\&(C_e \rightarrow fromEnv?[notify\_inCharge(s,o)])$

$not\_active\_obligation$: $\neg fromEnv?[notify\_inCharge(s,o)]$

The Two formulae describing the obligation rule are:

$$F3: context_e \rightarrow obligation$$
$$F4: \neg context_e \rightarrow not\_active\_obligation$$

Note that the provided LTL formulae respect the notation of the SPIN model checker. The parameters s, s', o and o' has to be instantiated with the values defined by the mapping in Table 5.2. It is possible to use a script that automatically assigns s, s', o and o' the values that fits the mapping.

	O2O	Functional Model	LTL
Security rule action	notify_Edit	?notify_inCharge(s,o) !notified(0,0)	fromEnv?[notify_inCharge(s,o)] toEnv?[notified(0,0)]
Security rule subject	*system*	message parameter s	message parameter s
Security rule object	O	message parameter o	message parameter o
State based context	-	-	$\top$
Event based context action	edit_file	?edit_file(s',o') !edit_grant(0,0)	fromEnv?[edit_file(s',o')] toEnv?[edit_grant(0,0)]
Event based context subject	S'	message parameter s'	message parameter s'
Event based context object	x.pdf	message parameter o'	message parameter o'

TABLE 5.2: The Mapping Table.

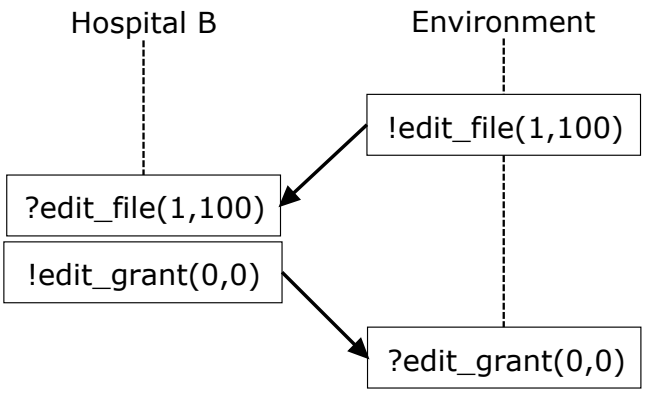


FIGURE 5.3: The Counter Example Generated by SPIN

5.4.4 Verification of Correctness

The generated LTL formulae are used to define *claims* in the SPIN model checker. The *claims* represent properties that should be satisfied during the execution of a model. We verify that all executions of the model are accepted by these *claims*. If the execution of the claim does not match with the execution of the model, SPIN will produce a *counter example* that allows the execution to be replayed. To show the applicability of our method we injected some faults in the functional model. For instance we investigate the case where the behavior of the obligation rule (specified in the above subsection) is removed from the functional model. Thus, after a medical file of a patient is edited by a doctor of hospital A the system does not notify the doctor in charge of this patient.

When running the verification process a counter example is generated by SPIN when checking the formula $O2$. This counter example is given in Figure 5.3. We also verified whether the obligation action actually notify the right doctor by verifying if the value of o corresponds to the doctor in charge of the medical report represented in o'.

5.5 Conclusion

In this chapter, we presented a model checking based approach for verifying the correctness of a system model with respect to interoperability security policies. The security policies are initially specified using O2O model. This model allows us of specifying unambiguously and concisely the security policies. The syntax of this model cannot be directly used in our verification. Thus, we first propose an approach to describe O2O security rules to the well known Linear Temporal Logic. We show that an O2O security rule can be described using two LTL formulae. In order to instantiate an LTL formula from a set of O2O security rules, we provided a mapping between the elements of the O2O security rule and the elements of the LTL formulae. Finally, we carried out a case study on a hospital network to show the applicability of our method.

As perspectives, the result of this study, mainly the description of O2O security policy with LTL, can be adapted for monitoring testing. Thus, the traces of two systems that communicate according to an interoperability security policy will be collected. These traces will be checked with respect to the LTL formulae requirements, and automatically a verdict of conformance of the implemented interoperability security policy with its requirements will be emitted. This will enable us to detect on real-time system crashes and security rules violations and most importantly to be able to stop this kind of malicious behaviors without any delay. Another possibility is to use the model checker to produce active test cases covering the requirements by using the counter-example provided by SPIN. In this case, the claims provided to SPIN will become a non desirable property to be checked.

In the Next chapter we discuss the both passive and active approaches for testing security policies. We present for each one the advantages and the drawbacks. Finally, a new method for testing security policies is proposed.

Part IV

Testing Interoperability Security Policies

"All men can see these tactics whereby I conquer, but what none can see is the strategy out of which victory is evolved."
Sun Tzu

Chapter 6

Testing Interoperability Security Policies

Contents

6.1 Introduction

Nowadays, we cannot dissociate the functional aspect of a system from its security considerations. Security policies specify the desired behaviors of a system when users perform actions on resources. Moreover, it is possible for a security policy to add new behaviors to the system such as obligation actions. In general, such obligation actions are not supported by the system. Therefore, testing only the functional parts of a system is not sufficient to guarantee that a system behaves as required and provides the intended services. This testing task has to be completed with a security testing phase.

In most cases, testing is based on the ability of a tester that simulates the implemen-

tation under test, in short IUT, and checks the correction of the answers provided by the implementation. It is worth to point out that in recent years, there has been a new emphasis on testing local security policies [MOC+07, MBCBN08, HXCL12, TG11]. These work can be classified into two approaches: *active testing* and *passive testing*. The two approaches are detailed in Chapter 3. Each of these approaches has drawbacks that limit its application for testing interoperability security policies. In this chapter, we discuss the drawbacks of these two approaches. Then we introduce a framework for testing interoperability security policies. We recall that in this work we consider OrBAC security policies, we use O2O to deal with access control in an interoperability context.

6.2 Motivation

In this section we discuss the process as well as the drawbacks of applying active or passive testing techniques to check the behavior of systems with respect to its interoperability security policies.

6.2.1 Limitations of the Active Testing

The main difference between testing local and interoperability security policy is that the latter involves a distributed system. Therefore, the existing methods to test local security policies has to be adapted to fit this new environment.

The key idea in active testing is that tester can interact, by providing inputs, with the implementation under test. That is in active testing the IUT is is modeled using formal description. This model is used to generate test suites. The strength of active testing is the ability to focus on particular aspects of the IUT, i.e. specific security requirements. In this section we present our approach to apply active testing to check interoperability security policy in an *informal* way. In particular, we present the possible test architectures that can be used for testing interoperability security policies. It is important to mention that this section does not provide details on the test case generation. This study aims to show the limitations of applying such methodology.

It is important to mention that in our study, we assume that interoperability security policies are always consistent and decision complete. In particular, we say that a security policy is consistent if it computes at most on decision for each incoming request, and it is complete if for any request the policy computes at least one decision. In this section, we present the active testing approach to check the interoperability security policies. Finally, we show the limitations of the active approach for testing interoperability security policies.

6.2.1.1 Test Case Generation and Test Verdict

In this Section we present the test case generation for interoperability security policies. Our main goal is to generate test cases to stimulate the O-grantee to send specific requests to the O-grantor. These requests target the security rules to be tested. We assume that we

have only access to the O-grantee system. For each request sent from the O-grantor one and only one security rule can be involved[1]. After applying the test case, we check that the set of received outputs conforms with the test case only if the conditions of context are satisfied.

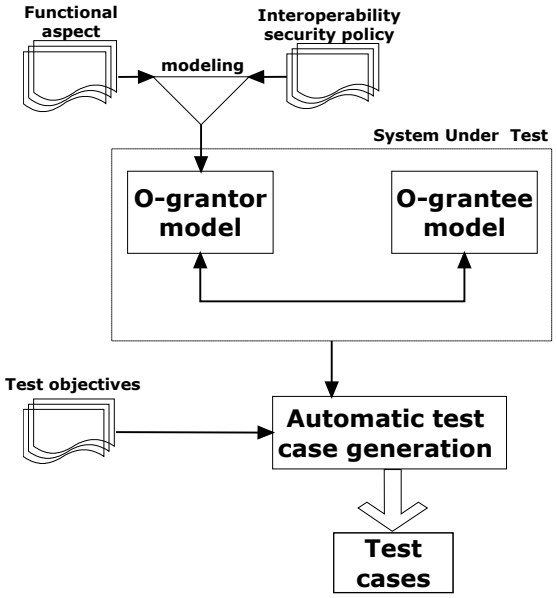


FIGURE 6.1: Overview of the Test Generation Framework.

Graphically, it is represented in Figure 6.1. The process of "Automatic test case generation" receives two inputs parameters. On the one hand the specification of the system by using a formal model. This model represent the joint behavior of the two systems (the O-grantor and the O-grantee). Let us note that the O-grantor model describes the functional aspect of the system and the interoperability security policy. On the other hand the test objectives specifying the security properties to be tested. Regarding the hole approach, these are the main concepts that allow us to describe it in detail:

1. *The test architecture*,
2. *the systems behaviors using a formal specification language*,
3. *the test objectives and the test case generation*, and
4. *the test execution and test verdict.*

Let us focus in the first topic 1, by means the test architecture. Our system consists of two systems: a system that represents the O-grantee and the other one that represents the O-grantor. On the one hand, the access is done by the users of the O-grantee so a control point should be placed at the user side of the O-grantee that initializes the interaction between the two systems. On the other hand a security rule cannot be tested unless it is active. To resolve this issue, we propose two solutions. The first one need another control point on the O-grantor. This control point will interact with the O-grantor in

[1]Let us remark that the interoperability security policy is consistent and decision complete.

order to activate the security rules. Figure 6.2 presents the test architecture. In the second solution, we need a compulsory minimum set of points of observation to attach to the architecture, but we are allowed to define higher number of points of observations where the exchanged data can be collected at runtime.

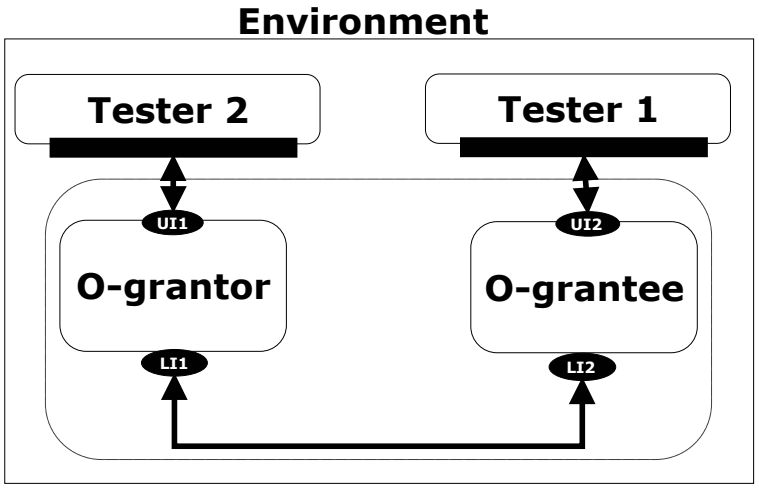


FIGURE 6.2: Active Testing Architecture With Two Tester.

Let us note that in our approach it is necessary to attach the following points of observations: those points that allow us to collect the interactions of the O-grantor with its information system which in charge of managing the security policies contexts, and those interactions between the O-grantor and the O-grantee when the information of a context are managed by the O-grantee. The test architecture is presented in Figure 6.3.

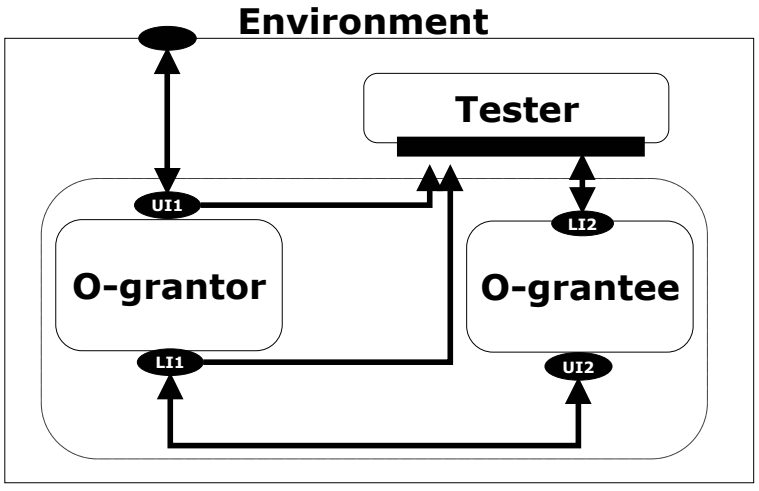


FIGURE 6.3: Active Testing Architecture With one Testers.

Now, let us focus on 2, that is the systems behaviors using a formal specification language process. The description shows the behavior of the communicating system by taking in consideration the test architecture, the interoperability security policy, and the different actions that intervene in the correct behaviors of the system. Each system behavior is described with a formal model. Let us note that the extended automaton that covers the

joint-behavior of the two systems can be built from the systems models.

Next we present the following concept, by means 3: the test objectives and test generation. This task is based on selecting a set of tests according to a given criteria. These criteria are defined with respect to the security requirements to be tested. A test objective as a list of ordered conditions. This means that the conditions have to be satisfied in the order that is given by the test objective. The test case generation is guided by the set of these test objectives. The inputs of the generated test case stimulate the system O-grantee that sends specific requests to the system O-grantor. Whereas, the outputs of the test cases represent the expected behavior of the system when the security rules hold. Let us remark that regarding the verdict, when we apply the inputs of the test case on the O-grantor system, the generated set of outputs has to be conform with respect to the test. Let us remark that a rule can be only checked when the rule holds. Thus, in the case where we adopt the test architecture described in Figure 6.2, a generated test case consists of two sets of inputs. A first set applied at the tester 2, it consists at activating the security rule to be tested. The other set consist of the inputs that are applied on the O-grantor. In the case where we adopt the test architecture presented in Figure 6.3, if the security rule is constrained by a context, an additional information that gives the state of this context is needed in the tests. The test verdict is built based on the information on the security rule context that are collected on the point of observation and the behavior related to the execution of the test case. Following we present the test verdict after executing a test case.

$$\texttt{Verdict} = \neg(\texttt{Test case} \oplus \texttt{context})$$

It is worth to remark that the global verdict, that is the verdict after performing all the test cases, has the form of a conjunction of the verdicts produced for each test case. Therefore if the verdict of one test case is false the global verdict is also false.

6.2.1.2 Limitation of Applying the Active Testing

The main limitation when applying the test architecture described in Figure 6.2, is to be able to synchronize the two tester. The two tester need to coordinate in order first to activate a security rule, then apply the test case that checks this rule. Thus, if the testers are not synchronized this sequence could be broken and the test case will not be valid.

The architecture presented in Figure 6.3 has a similar drawback. In this case the outputs that are received form the different points of observation has to be properly correlated with the output generated in the control point. For the both cases, the synchronization parts is not a simple task as described in [LDB+93].

6.2.2 Limitation of the Passive Testing

Passive testing consists in observing the interaction between the O-grantor and the O-grantee systems in run-time. It does not disturb the natural run-time of these systems, that is why it is called passive testing. The record file of the event observation is called an

event trace. This trace will be analyzed according to the interoperability security policy specification in order to determine the conformance relation between this trace and the specification.

The main drawback of the passive testing is that we can only check those behaviors that appear in the collected trace. Thus, depending on the length of the trace, it is probable that some of the security requirements cannot be verified since the behavior related to these security requirements may not appear in the trace. Therefore, we should choose a longer trace in order to verify these security properties. Otherwise, the test verdict of these security properties is *inconclusive*. Note that when choosing a longer trace the complexity of applying the passive testing algorithm will grow higher.

6.3 Testing Interoperability Security Policies

In order to overcome the drawbacks of the passive and active testing approaches, we propose to combine the two approaches in such a way to gain from both the advantages and eliminate the drawbacks [EMCA12].

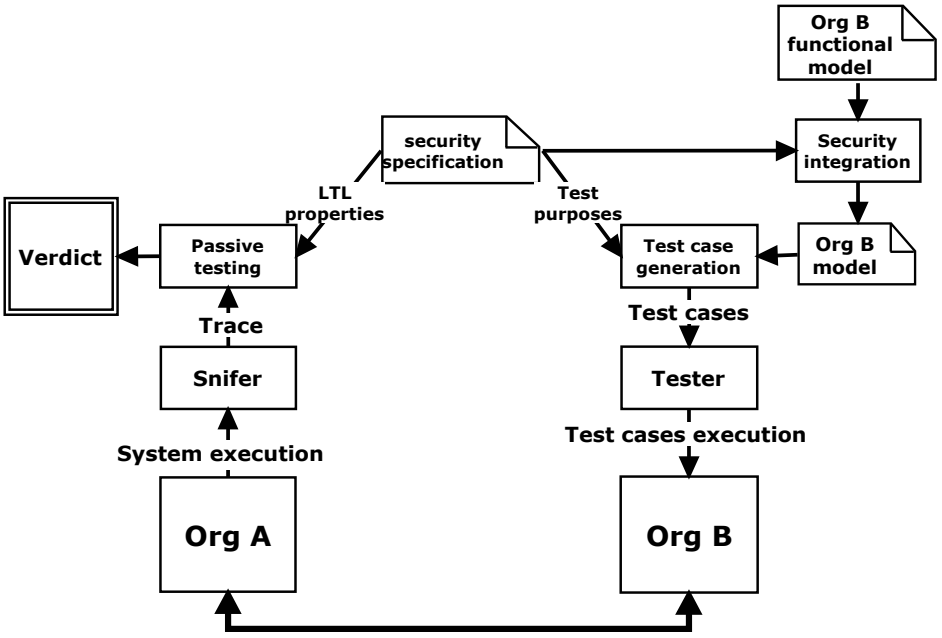


FIGURE 6.4: Contextual Interoperability Security Policy Testing.

In Figure 6.4, we illustrate the overall approach. We consider the interaction between only two organizations. The OrgB is considered the O-grantor organization. This organization hosts the resources to be accessed by the the O-grantee organization (OrgA). In the case where more than two organizations are involved, the same procedure is repeated for each pairs of O-grantee and O-grantor.

- **Step 1 (the active testing part):** The first step of this approach is to generate test cases to stimulate the OrgA system to send specific requests to the OrgB system.

These requests target the security rules to be tasted.

- **Step 2 (the passive testing part):** We collect the executions of OrgB after applying the test cases. The collected trace contains the necessary information to check whether the the behsvior of the organization OrgB conforms with its interoperability security policy. In this step we rely on passive testing techniques.

Note that the main objective of the Step 1 is to construct a test suit that will stimulate the O-grantor to perform specific behavior. These behaviors will be collected in an execution trace in the Step 2. It is important to mention that by constructing the trace execution using the test suit, this trace will contain all those behaviors that we need to test. In addition, the O-grantor system will not be disturbed during the testing process since we only observe its behaviors.

In the following we illustrate the steps 1 and 2. We also provide use cases to show the applicability of our work.

6.4 Step 1: Test Case Generation for Interoperability Security Policies

6.4.1 Test Generation Methodology

Our aim in this part is to generate test cases to stimulate the O-grantee to send specific requests to the O-grantor.

The process of test case generation is similar to the explained in Figure 6.1. It is important to mention that in this step we do not consider the activation of a security rule, since this issue will be included in the Step 2 of our testing process. Thus, the test architecture contains only one tester on the user side of the O-grantee. In particular, the tester simulates the behaviors of the users of the O-grantee to initialize the interaction between the two systems.

The communicating system behavior and the interoperability security policy are modeled with extended finite automata. We can model the joint-behavior of the two organizations with an extended automaton and construct its reachability graph, which is a directed graph or a transition diagram. The extended automaton that covers the joint-behavior of the two systems is constructed from the extended automata of these systems.

As the size of the reachability graph that result from this cartesian product is in general large, the use of traditional methods that produce a global reachability graph from which tests are generated is not possible. Therefore, it is necessary to use methods based on partial generation of the reachability graph by applying different methods [CLRZ99]. In these methods test case generation is guided by the test architecture and a predefined set of test objectives. Different tools can be used for test case generation (TestGen-IF, SIRIUS, TestComposer). We choose to use the TestGen-IF tool which is an open source tool that accepts systems modeled with the IF language. This language allows to formally describe a communicating system as active processes instances running in parallel and interacting

asynchronously through shared variables or signals which can be send via signal routes or by direct addressing. Each process is modeled as an extended finite automaton. The tool also allows to constructs tests sequences with high fault coverage and avoids the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random search.

The model of a system is the abstraction of a real implementation. In this model, irrelevant details are abstracted to leave a simplified description of the system behavior. In order to execute the test cases generated with TestGen-IF in a real system, it is mandatory to instantiate them. The instantiation of an abstract test case is composed of two sub-processes: the concretization (addition of details) and the translation to executable scripts. The final test case is a script that contains details of the implementations such as the real values of variables and the signals of the test case. This script is executed in the O-grantee system to stimulate this system generate specific request to the O-grantor.

6.4.2 The Hospitals Network Case Study

We present the test case generation approach through the hospital network case study presented in Chapter 4. We recall that our aim is to check the interoperability security policy of hospital B. Thus, we consider only the one way of interaction, from hospital A to hospital B. Thus, the resources to be accessed are located in hospital B. We will index all the roles that comes from hospital A with the letter A. The VPO in charge of the interaction is denoted by VPO_{A2B}.

We also assume that each hospital has its local security policy to manage the privileges of its local users. We assume that the following *roles* exist in the two hospitals: doctor, nurse, Aduser and ITuser. It is also assumed that in both hospitals a patient has a medical report, data related for payment and sensitive data which are related to personal information (previous medical report, insurance company, etc.). Also in each hospital we have some system files related to its information system.

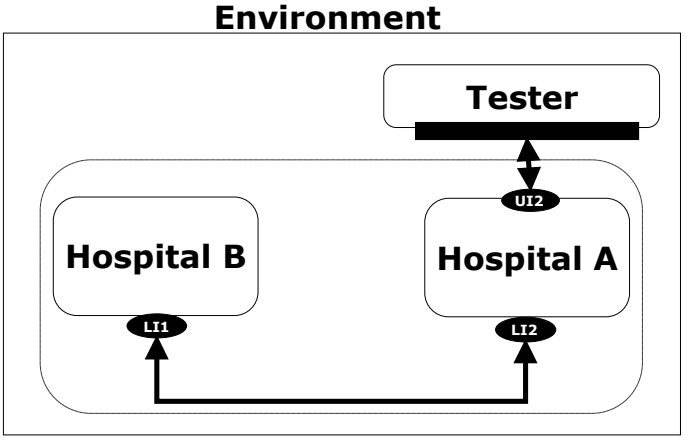


FIGURE 6.5: Active Testing Architecture.

6.4.2.1 The Formal Model

The model of hospital A and hospital B are represented using the IF language [BGM02]. The interoperability security policy of hospital B is integrated in its model based on the process described in Chapter 4. The behavior of this model is now constrained by the interoperability security policy. We also verified the correctness of our integration process by model checking.

6.4.2.2 Test Case Generation Using TestGen-IF

In this section we will present the automatic test case generation process. In Figure 6.5 we present the test architecture.

The test cases are generated by using an automatic test case generation tool, TestGen-IF [COML08]. The tool is based on the IF-2.0 simulator [BGO+04] that allows to construct an accessibility graph from an IF specification. It implements an automated test generation algorithm based on a Hit-or-Jump exploration strategy [CLRZ99] and a set of test purposes. In Figure 6.6 we show the basic architecture of the TestGen-IF tool.

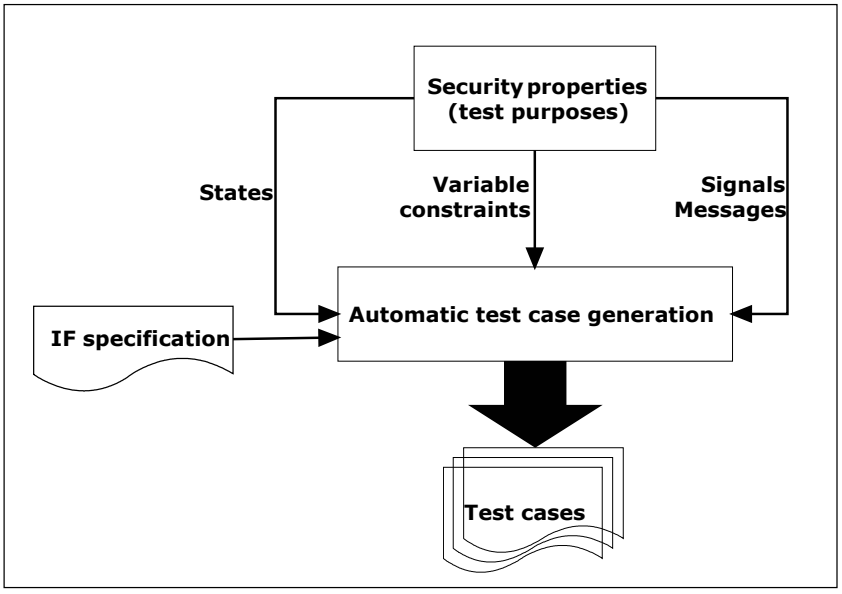


FIGURE 6.6: The Basic Architecture of TestGen-IF.

From the current state, the tool constructs a partial accessibility graph. The partial graph is constructed using breath first search or depth first search algorithm. The depth of the partial graph can be specified in the tool inputs. At any moment it conducts a local search from the current state in a neighborhood of the partial graph. If a state is reached and one or more test objectives are satisfied (a Hit), the set of test objectives is updated and a new partial search is conducted from this state. Otherwise, a partial search is performed from a random graph leaf (a Jump). The advantages of TestGen-IF tool is that it efficiently constructs tests sequences with high fault coverage, avoiding the state explosion and deadlock problems encountered respectively in exhaustive or exclusively

random searches.

A test objective is a set of ordered conditions. This means that the conditions have to be satisfied with the order that is given by the test case. A condition is a conjunction of a process instance constraint, state constraint, action[2] constraints and variable constraints. A process instance constraint indicates the identifier of a process instance. An action constraint describes sending or receiving messages. A variable constraint gives conditions on variable values. A test objective specifies the property to be validated in the system implementation. The test case generation algorithm fetch for this property in the reachability graph that describes the joint-behavior of the two systems. When the property is satisfied, the algorithm generates the path that leads to this property. This path is transformed to an executable test case to be applied later to the system implementation.

In our test case generation process, we assume that the interoperability security policy is decision complete (or simply complete). Thus for each request the interoperability security policy computes one and only one decision. Our objective is to test those behaviors of hospital B that are generated after the execution of the interoperability security policy. In particular, we want to test whether these behaviors conform with the interoperability security policy. Thus, these behaviors constitute our test objectives. The following shows the test purpose which correspond to the rule that permits a doctor of hospital A to read sensitive information after signing a non divulgation form.

Objective

$$
\begin{aligned}
tp &\leftarrow \left\{ \bigwedge_{1 \leq i \leq 5} c_i \right\} \\
c_1 &\leftarrow (process : instance == \{hospitalB\}) \\
c_2 &\leftarrow (state : source == \text{“}sign\text{”}) \\
c_3 &\leftarrow (state : target == \text{“}idle\text{”}) \\
c_4 &\leftarrow (action : input == \text{“}access\_file\{6, 65\}\text{”}) \\
c_5 &\leftarrow (action : output == \text{“}access\_grant\text{”})
\end{aligned}
$$

To test the behavior of hospital B with respect to the interoperability security policy, we identified 53 test objectives. Table 6.1 gives some metrics on the test case generation process.

Test objectives	Partial graph generation	Depth limit	Visited states	Time to generate the test cases
53	BFS	30	12855	28 min and 2 secs

TABLE 6.1: Experimental Results on Test Case Generation.

Note that the test purpose constraints can only have a specific value. Thus, the test case generated by a test purpose corresponds to specific values of subject and object. Therefore,

[2] An action represents in this context an input or an output

to cover all the values of subjects and objects that are assigned to the role and the view which are defined in the security rule, this test case has to be applied several time each time with different values of subject and object.

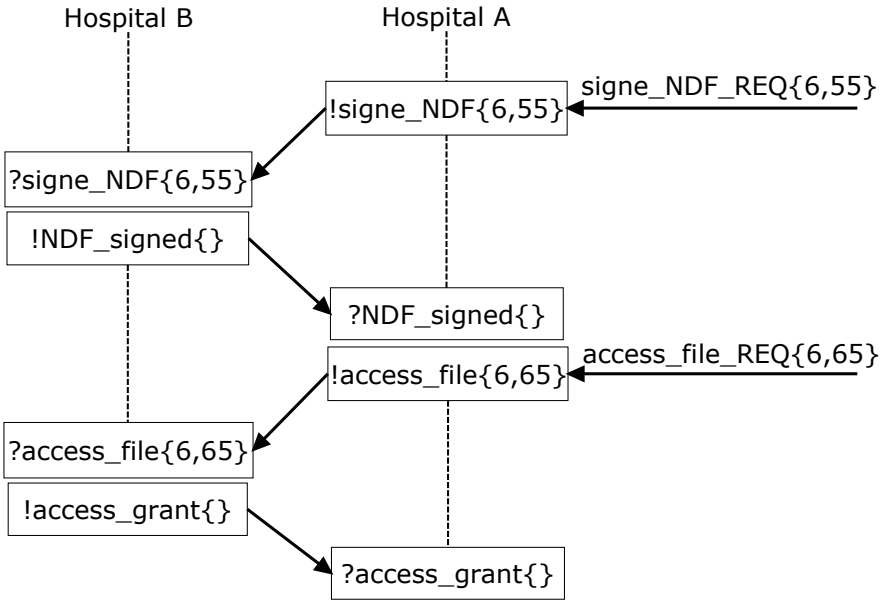


FIGURE 6.7: Example of a Test case Generated by TestGen-IF.

Figure 6.7 presents the test case generated by the above test objective.

6.5 Step 2: Passive Testing Approach

In the previous step, we generated the set of test cases to stimulate the system of OrgB. The collected traces from the system execution are used to check if this system behaves as required by the interoperability security policies. In this section, we present the algorithm and the architecture of the passive testing process. The test architecture shows the location of the probes that collect the event trace to be analyzed.

6.5.1 Preliminaries

We can distinguish four steps in our passive testing methodology for security checking:

- The definition of the passive testing architecture: in general to collect the execution trace. We need to install observation points (called also probes) into specific strategic points. These observation points aim to collect data exchanged between relevant entities.
- The definition of the security policy using a formal specification language: the description concerns the interoperability security rules that the studied system has to respect. We rely on the linear Temporal Logic (LTL) formalism to describe the security rules.

- The definition of the algorithm for security checking: the definition of the algorithm is based on the formal description of the security policies and the format of the execution trace. This algorithm is the core of the passive tester. It aims to check the security rules in the collected execution trace.
- The security analysis: the passive tester has to perform security analysis on the execution trace in order to deduce a global verdict. This verdict is *PASS* if the trace respect all the specified security rules and FAIL if it does not.

6.5.2 The Satisfaction Relation in LTL

Linear temporal logic (LTL) is a logic in which one can express properties of paths in a computation tree. In particular, it allows one to reason about both causal and temporal properties. The formalism is introduced in Chapter 5. In this section we present the satisfaction relation as defined in [GK05]. Let us consider a trace as a finite list of events $\{e_1, ..., e_n\}$. Assume two partial functions defined for non-empty trace $head : Trace \rightarrow event$ and $tail : Trace \rightarrow Trace$ for taking the head and the tail respectively of a trace. That is $head(e,t) = e$, $tail(e,t) = t$, $length(end) = 0$ and $length(e,t) = 1 + length(t)$ where t is a trace, e is an event and end denotes the empty trace. We further assume that for any trace t, t_i denotes the suffix that starts at position i where i is a natural number.

Definition 6 *The satisfaction relation* $\models \subseteq Trace \times Formula$ *which define when a trace* t *satisfies a formula* ϕ *(written* $t \models \phi$*) is defined inductively over the structure of the formula as follows (where* p *is an atomic proposition and* ϕ_1 *and* ϕ_2 *are any formulae):*

$t \models true$	iff	$true$
$t \models flase$	iff	$false$
$t \models p$	iff	$t \neq end$ *and* $head(t) = p$
$t \models \neg p$	iff	$t \not\models p$
$t \models \phi_1 \wedge \phi_2$	iff	$t \models \phi_1$ *and* $t \not\models \phi_2$
$t \models \phi_1 \vee \phi_2$	iff	$t \models \phi_1$ *or* $t \models \phi_2$
$t \models \phi_1 \rightarrow \phi_2$	iff	$t \not\models \phi_1$ *or* $t \models \phi_2$
$t \models \Box\phi_1$	iff	$\forall i \leq length(t)\ t_i \models \phi_1$
$t \models \Diamond\phi_1$	iff	$\exists i \leq length(t)\ t_i \models \phi_1$
$t \models \bigcirc\phi_1$	iff	$t \neq end$ *and* $tail(t) \models \phi_1$
$t \models \phi_1 \cup \phi_2$	iff	$(\exists i \leq length(t))(t_i \models \phi_2$ *and* $(\forall j < i)t_j \models \phi_1)$

6.5.3 Passive Testing Methodology

In this section we describe our passive tester tool. Its architecture is presented in Figure 6.8. The security test tool receives two inputs. The security specification modeled by using LTLs and the execution trace to be analyzed. The trace is firstly analyzed through a pre-processing phase that performs the following tasks:

- Filtering the trace: The basic idea is to keep in the trace only relevant information to the security properties to be checked.

- Parsing the global trace: in this step a list of events is created from the filtered trace. This event list constitutes the input to our passive testing algorithm. Each event of this list corresponds to an emission or a reception of a message between the supervised entities.

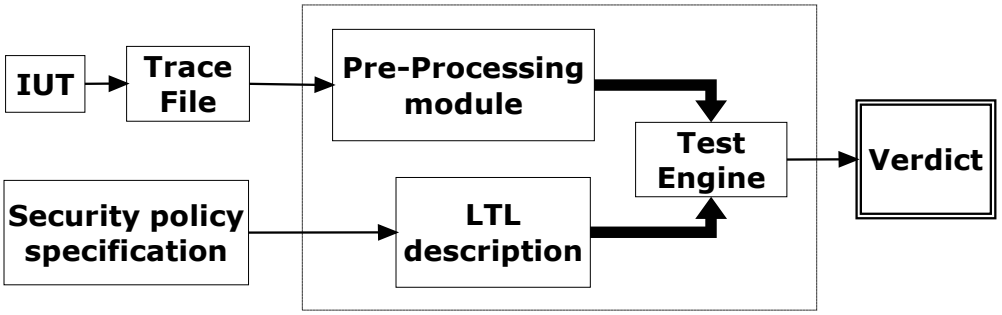


FIGURE 6.8: Passive Testing Methodology.

Finally, the trace is analyzed using the passive testing algorithm. It checks this trace verifies security requirements.

6.5.4 Trace Specification

In this work we are dealing with "off-line" testing using a pre-collected trace after stimulation of the system that implements the interoperability security policy to be verified. Therefore, the collected trace is considered finite. The trace must contain enough information to passively test the interoperability security policy. This information could be for instance a request, the sender of this request, the object to be accessed, the access decision, etc. In the case where the information in the trace are not sufficient to verify some specific security requirements, additional information has to be collected by installing additional probes to the system under test.

6.5.5 Passive Testing Algorithm

In this section we present the passive testing algorithm for verification and validation of security properties. The inputs of the algorithm are a set of security requirements specified in LTL and the collected trace. The aim is to provide a verdict about the conformance of the trace with respect to the specified security properties. The algorithm checks whether the trace verifies the LTL formula. The algorithm generates a FAIL verdict if this formula is not respected in the trace. The test algorithm is based on the idea that LTL properties can be checked backwards by updating the verdict at each step based on our knowledge of the future (as each line of the trace table is traversed from its end). We will first start by an using example of a simple LTL formula to show how it can be checked on a trace. Consider, for instance, the following:

$$\phi = \Box(P \rightarrow \Diamond Q)$$

The Breath First Search order of this formula gives the following set of sub-formulae.
$\phi_1 = \Box(P \rightarrow \Diamond Q)$
$\phi_2 = P \rightarrow \Diamond Q$
$\phi_3 = P$
$\phi_4 = \Diamond Q$
$\phi_5 = Q$

Now consider a finite trace $Tr = \{e_1, ..., e_i, ..., e_n\}$. One can define recursively a boolean matrix $mat[1..n, 1..m]$ where n is the length of Tr and m is the number of sub-formulae with the meaning that $mat[i, j] = true$ iff $Tr_i \models \phi_j$. in our example it will be $mat[1..n, 1..5]$ such as. $mat[i, 5] := (Q \in e_i)$
$mat[i, 4] := mat[i, 5] \vee mat[i + 1, 4]$
$mat[i, 3] := (P \in e_i)$
$mat[i, 2] := mat[i, 3] \rightarrow mat[i, 4]$
$mat[i, 1] := mat[i, 2] \wedge mat[i + 1, 1]$
for all $i < n$, where $\wedge, \vee, \rightarrow$ are ordinary boolean operators. For $i = n$, we need to initialize the matrix as follows. $mat[n, 5] := (Q \in e_n)$
$mat[n, 4] := mat[n, 5]$
$mat[n, 3] := (P \in e_n)$
$mat[n, 2] := mat[n, 3] \rightarrow mat[n, 4]$
$mat[n, 1] := mat[n, 2]$

An important observation is that, for each element of Tr, we may need at worst information about the previous element (the next one when addressed backwards). Therefore, instead to keep all the table $mat[1..n, 1..m]$ which would be quite large in practice, one needs only to keep rows $mat1[i, 1..m]$ and $mat2[i+1, 1..m]$ handling the information about actual step and the next one. We will call these vectors V_{now} and V_{next} respectively. Algorithm 3 presents the passive testing algorithm. Given an LTL formula ϕ and for each line of the trace table, this algorithm consist of three main phases:

1. First we generate the set of sub-formulae in the BFS order of the tested LTL formula. Let $\{\phi_1, ..., \phi_i, ..., \phi_m\}$ be the list of all generated formulae. The semantics of finite LTL trace allows us to determine the truth value of $TR_i \models \phi_j$ from the truth value of $Tr_i \models \phi_{j'}$ for all $j < j' \leqslant m$. This recurrence justifies the backward checking order of the algorithm.
2. The second step is an initialization loop. Before the main loop, we should first initialize the vector $V_{next}[1..m]$. According to the semantic of LTL, the vector V_next is filled backwards. For a given $1 \leqslant j \leqslant m$, $V_next[j]$ is calculated as follows:
   - If ϕ_j is a variable then $V_{next}[j] \rightarrow (\phi_j \in e_n)$; Here we only verify if the atomic proposition satisfies the last event from the trace;
   - If ϕ_j is $\neg\phi_{j'}$ for some $j < j' \leqslant m$ then $V_{next}[j] \rightarrow not\ V_{next}[j']$ where not is the negation operator on boolean;
   - If ϕ_j is $\phi_{j_1} Op \phi_{j_2}$ for some $j < j_1$, $j_2 \leqslant m$ then $V_{next}[j] \rightarrow V_{next}[j_1] op V_{next}[j_2]$, where Op is any propositional operator and op is its corresponding boolean operation;

Algorithm 3: Checking LTL properties

input : An LTL formula ϕ and an event execution trace $Tr = \{e_1, ..., e_n\}$
output: A verdict about the conformance of ϕ w.r.t. Tr

Generate a set of sub-formulae in BFS order $(\phi_1, ..., \phi_m)$
/ Initialization */*
for $j = m \rightarrow 1$ **do**
  if ϕ_j *is a variable* **then**
    $V_{next}[j] := (\phi_j \in e_n)$;
  if $\phi_j == \neg\phi_{j'}$ **then**
    $V_{next}[j] := (not\ V_{next}[j'])$;
  if $\phi_j == \phi_{j_1}\ Op\ \phi_j 2$ **then**
    $V_{next}[j] := (V_{next}[j_1]\ op\ V_{next}[j_2])$;
  if $\phi_j == ((\bigcirc\phi_{j'}) \parallel (\Diamond\phi_{j'}) \parallel (\Box\phi_{j'})$ **then**
    $V_{next}[j] := V_{next}[j']$;
/ Main Loop */*
for $i = n - 1 \rightarrow 1$ **do**
  for $j = m \rightarrow 1$ **do**
    if ϕ_j *is a variable* **then**
      $V_{now}[j] := (\phi_j \in e_n)$;
    if $\phi_j == \neg\phi_{j'}$ **then**
      $V_{now}[j] := (not\ V_{now}[j'])$;
    if $\phi_j == \phi_{j_1}\ Op\ \phi_j 2$ **then**
      $V_{now}[j] := (V_{now}[j_1]\ op\ V_{now}[j_2])$;
    if $\phi_j == \bigcirc\phi_{j'}$ **then**
      $V_{now}[j] := V_{next}[j']$;
    if $\phi_j == \Box\phi_{j'}$ **then**
      $V_{now}[j] := V_{next}[j] \wedge V_{now}[j']$;
    if $\phi_j == \Diamond\phi_{j'}$ **then**
      $V_{now}[j] := V_{next}[j] \vee V_{now}[j']$;
    if $\phi_j == \phi_{j_1} \cup \phi_{j_2}$ **then**
      $V_{now}[j] := V_{now}[j_2] \vee (V_{next}[j_1] \wedge V_{next}[j_2])$;
  $V_next := V_{now}$;
$Verdict := V_{next}[1]$;

- If ϕ_j is $\bigcirc\phi_{j'}$, $\Box\phi_{j'}$ or $\Diamond\phi_{j'}$ then clearly $V_{next}[j] \rightarrow V_{next}[j']$ due to the stationary semantics of the finite trace LTL;
- ϕ_j is $\phi_{j_1} \cup \phi_{j_2}$ for some $j < j_1,\ j_2 \leqslant m$ then $V_{next}[j] \rightarrow V_{next}[j_2]$ for the same reason as above.

3. The last step is the main loop. The trace Tr is visited backwards, the loop index will vary from $n-1$ to 1. The loop body will update the vector V_{next} to serve as basis for the next iteration. At a certain iteration i, the vector V_{now} is updated backwards as follows:
   - If ϕ_j is a variable then $V_{now}[j] \rightarrow (\phi_j \in e_n)$;
   - If ϕ_j is $\neg\phi_{j'}$ for some $j < j' \leqslant m$ then $V_{now}[j] \rightarrow not\ V_{now}[j']$;
   - If ϕ_j is $\phi_{j_1} Op \phi_{j_2}$ for some $j < j_1,\ j_2 \leqslant m$ then $V_{now}[j] \rightarrow V_{now}[j_1] op V_{now}[j_2]$, where Op is any propositional operator and op is its corresponding boolean operation;
   - If ϕ_j is $\bigcirc\phi_{j'}$ then $V_{now}[j] \rightarrow V_{now}[j']$ since ϕ_j holds now iff $\phi_{j'}$ held at the previous step (which processed the next event, the $(i+1)^{th}$);
   - If ϕ_j is $\Box\phi_{j'}$ then $V_{now}[j] \rightarrow V_{now}[j'] \wedge V_{next}[j]$ because ϕ_j holds now iff $\phi_{j'}$ holds now and ϕ_j held at the previous iteration;
   - If ϕ_j is $\Diamond\phi_{j'}$ then $V_{now}[j] \rightarrow V_{now}[j'] \vee V_{next}[j]$ for similar reason as above;
   - If ϕ_j is $\phi_{j_1} \cup \phi_{j_2}$ for some $j < j_1,\ j_2 \leqslant m$ then $V_{now}[j] \rightarrow V_{now}[j_2] \vee (V_{now}[j_1] \wedge V_{next}[j])$.

After each iteration $V_{next}[1]$ says whether the initial LTL formula is validated by the trace Tr. Therefore desired output is $V_{next}[1]$ after the last iteration. The truth value of this vector element gives the final verdict ($true \equiv PASS$ and $false \equiv FAIL$). The analysis of this algorithm is straightforward. Its complexity is $O(n.m)$ where n is the length of a line of the trace table and m is the number of sub formulae generated from the LTL formula in the BFS order.

6.5.6 Describing OrBAC Security Rules with LTL

In our study, we consider OrBAC security policies. We discussed in Chapter 5 that the OrBAC formalism cannot be directly used for the verification and validation of a security policy. Therefore, we suggested describing the security rules using the formal language LTL. In Chapter 5, we used this formalism to verify the correctness of interoperability security rules which are integrated in the functional model of a system. This work can be useful in this study. The only difference is that in the case of a permission or a prohibition rule, the informations related to the security rule context are considered delivered with the request to the Policy Decision Point (PDP). In our testing process, we do not differentiate between event based and state based context. Thus, a permission or a prohibition rule can be described with the LTL formalism with the following two formulae:

$$F1:\ Context \rightarrow (Subject \wedge Action \wedge Object \rightarrow (\neg unexDecision \cup exDecision))$$
$$F2:\ \neg Context \rightarrow (Subject \wedge Action \wedge Object \rightarrow (\neg exDecision \cup unexDecision))$$

where *Subject*, *Action*, *Object* and *Context* are formulae that describe a subject of a role, an action of an activity, an object of a view and the context of the security rule respectively. Let us note that an implementation of a security policy can calculate several possible evaluation results such as *Permit*, *Deny* or *Indeterminate*. *Permit* or *Deny* are typically returned. For instance, in XACML if some required attribute is missing, *Indeterminate* is returned, with the *missing-attribute* status. Therefore, the formula *exDecision* verifies that when the security rule is activated, the decision of the implemented security policy is as expected by specification. The formula *unexDecision* checks if an unexpected decision is generated by the security policy. An obligation rule specifies that a role is obligated to perform an activity on a view. This obligation is activated when a specific action is performed on the system. We describe an obligation security rule in the LTL formalism with the following formulae:

$$F3: \ Context \rightarrow Obligation$$
$$F4: \ \neg Context \rightarrow \neg Obligation$$

The formula *Obligation* verifies that the action of the obligation rule is performed. The formula *Context* verifies that the action that triggers the obligation rule is executed. This formula can be detailed as follows:

$$Context : Subject \wedge Action \wedge Object \rightarrow (\neg unexDecision \cup exDecision)$$

This formula means that if a specific decision is generated after a specific role request to perform a specific activity on a specific view, the obligation rule will be activated and executed on the system.

6.5.7 Application to a XACML Framework

In this section we show how we fit our testing process into an authorization standard such as XACML (eXtensible Access Control Markup Language). XACML is an XML-based language for access control that has been standardized in OASIS. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies. The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses). This standard is used to implement the interoperability security policy in the system of hospital B of our case study.

6.5.7.1 The XACML Framework

In a typical XACML usage scenario, a subject (e.g. human user, workstation) wants to take some action on a particular resource. The subject submits its query to the entity protecting the resource (e.g. filesystem, web server). This entity is called a Policy Enforcement Point (PEP). The PEP forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information. The PEP then sends this request to a Policy Decision Point (PDP), which examines the request, retrieves

policies (written in the XACML policy language) that are applicable to this request, and determines whether access should be granted according to the XACML rules for evaluating policies. That answer (expressed in the XACML response language) is returned to the PEP, which can then allow or deny access to the requester. XACML has many benefits over other access control policy languages:

- One standard access control policy language can replace dozens of application-specific languages.
- Administrators save time and money because they do not need to rewrite their policies in many different languages.
- Developers save time and money because they don't have to invent new policy languages and write code to support them. They can reuse existing code.
- Good tools for writing and managing XACML policies will be developed, since they can be combined with many other applications.
- XACML is flexible enough to accommodate most access control policy needs and extensible so that new requirements can be supported.
- One XACML policy can cover many resources. This helps to avoid inconsistent policies on different resources.
- XACML allows one policy to refer to another. This is an important issue for large organizations. For instance, a site-specific policy may refer to a company-wide policy and a country-specific policy.

The OASIS also defined the concept of profile of XACML, that illustrates how organizations, that would like to use the RBAC model, can express their access control policies within this standard language. The XACML standard specification version 1.1 is implemented by SUN[3]. The implementation provides useful APIs and it describes how to use the APIs to build and customize PDPs, PEPs, or any related modules that fit into the XACML framework. Nevertheless, the RBAC profile of XACML shows some limitations that respond to all the requirements of the expressive OrBAC model.

In [AHCBCD06] the authors presented an extended RBAC profile to meet the OrBAC requirements. This extension is depicted in Figure 6.9. The policies are written by the PAP (1). When the PEP receives a concrete request (2), it is relayed to the context handler (3). The context handler collects the assignments of each subject to its roles, the resource to its views, and the action to its activities, and finally, gets the value of the contexts, by sending requests to the corresponding Enablement Authorities (4). Notice that the evaluation order does not matter. As depicted in Figure 6.9, the Enablement Authorities has 4 functionalities: REA, VEA, AEA, and CEA that respectively maintain a list of the defined roles, views, activities, and contexts values inside the organization. Each authority will make a request to the corresponding assignment policy for each candidate value. The CEA, and not the context handler in this model, should send additional requests to an information system that manages that kind of information. In this way, a part of the normal job of the XACML's context handler, in the evaluation of the request information, is relayed to this new entity. All the assigned values are sent back to the context handler

[3]http://sunxacml.sourceforge.net

(5). This latter will add the initial concrete values for each entity and send a request to the PDP for final evaluation (6). This request contains the abstract values and the concrete values of the subject, action, and object of the initial request in addition to the value of the context(s). The PDP evaluates the request according to the policies and sends back a response to the context handler (7) that will transmit it to the PEP in its native response language (8). The PEP may have to fulfil some obligations (9) before allowing or denying the access.

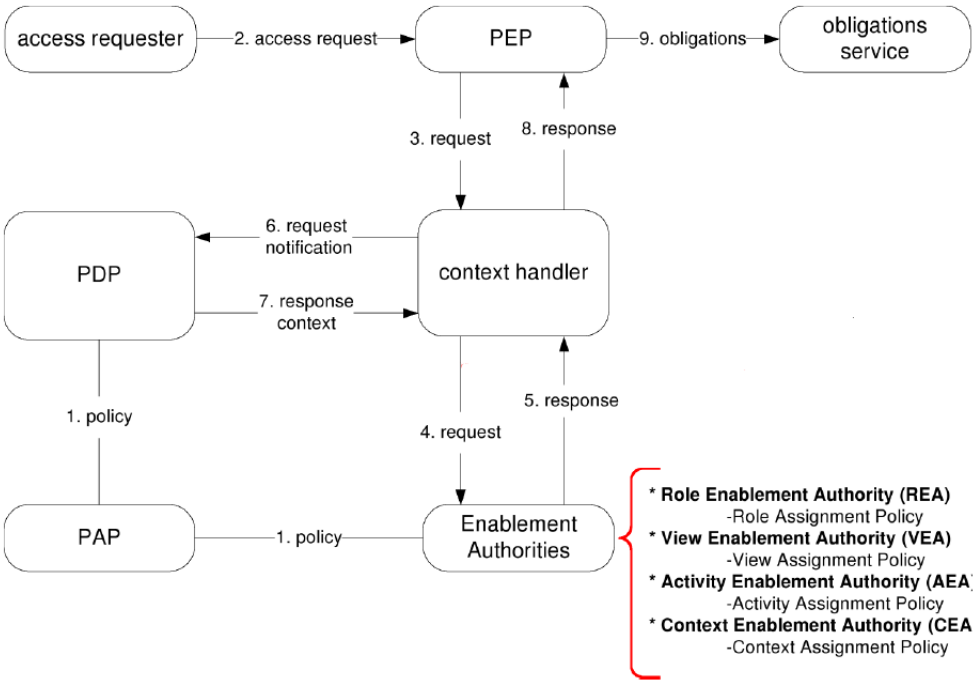


FIGURE 6.9: The Extended Profile Architecture.

The implementation of this profile is described and illustrated in [Hai08]. It is an extension of the existing SUN implementation. In this implementation, the environment value (as defined in the XACML standard) is never evaluated. This environment information is needed in the extended profile in order to carry the context value. That is, some modifications have been added to the existing classes in order to handle such information:

- The *PDPEnablement* class allows the creation of an XACML PDP that evaluates an XACML request against its corresponding policy. A PDPEnablement instance is created for each enablement authority and instantiated with the corresponding Assignment- Policy file. In this case, the PDPEnablement evaluates XACML enablement requests against XACML assignment policies. This class is also used to create the OrBAC PDP based on the same functionalities. The OrBAC PDP evaluates an XACML request including OrBAC requirements (i.e. roles, views, activities and contexts) against the general organizations OrBAC access control policy.
- The *ContextHandler* class implements the Context Handler entity. This entity manipulates a concrete XACML request made of subject, action and resource. This class has methods to extract the subject, action and resource from the request. Besides, the *getResourceContent* method allows to get the part of the accessed resource

that may be needed for the context evaluation. These extracted values are used by another method, i.e. *getRequestEnvironment*, that forges an XACML Subject entity whose attributes are those of the subject, action and resource of the initially received request. The returned set of the roles, activities, views and contexts are used by the context handler to generate an OrBAC request that is sent to the OrBACPDP for evaluation.

- The *RoleEnablementAuthority*(REA), *ActivityEnablementAuthority*(AEA), *ViewEnablementAuthority*(VEA) and *ContextEnablementAuthority*(CEA) classes implement respectively the Role, Activity, View and Context Enablement Authority entities. These classes manage the list of the roles, activities, views and contexts files defined within the organization.

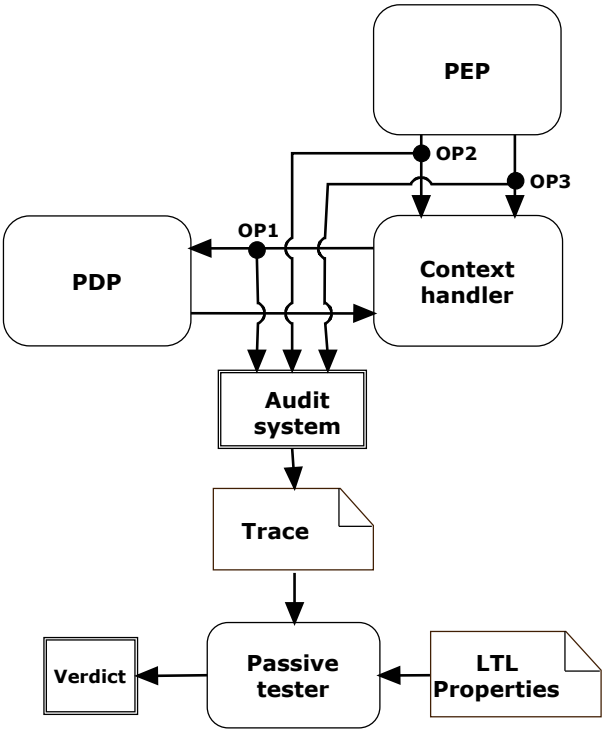


FIGURE 6.10: Passive Testing Architecture.

6.5.7.2 Test Architecture for the Passive Testing

In this section we present the test architecture for the passive testing step. This test architecture specifies the information that have to be included in our execution trace. It is important to mention that the trace is collected at the same time when applying the test cases that are generated in the step 1 of our testing process. Figure 6.10 presents our test architecture.

The trace constitutes of the requests and the responses which are respectively received and generated by the PEP. This trace is collected with the probes OP2 and OP3. We also collect the requests sent from the context handler to the PDP. This request serve to collect

information related to the context of a security rule. Next we show that the collected trace contains all the necessary information for testing security policies.

6.5.7.3 Security Rules Checking

In this section we present how we check whether a collected trace respect an interoperability security policy. The interoperability security policy is assumed XACML based and it is implemented as described in [Hai08].

LISTING 6.1: Example of a Request.

```
<Request>
<Subject>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
<AttributeValue>jsmith@users.example.com</AttributeValue>
</Attribute>
</Subject>
<Resource>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>report-Bob</AttributeValue>
</Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
<AttributeValue>read</AttributeValue>
</Attribute>
</Action>
</Request>
```

Let us consider that the following permission rule exists in the PDP: *users with a role attribute of physician of hospital A can perform the read action on a medical report in the context designated doctor.*

The PEP sends XACML requests to the context handler and receives XACML responses that it uses to grant or deny access. Listing 6.1 shows an XACML request in which the user, *jsmith@users.example.com*, is attempting to perform a read action on the medical report *report-Bob*.

LISTING 6.2: Example of a Response.

```
<Response>
<Result>
<Decision>Permit</Decision>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response>
```

Once the PDP receives the request, it processes the request against its policy. Because *jsmith@users.example.com*'s role attribute is *physician* and the context is satisfied, the resulting decision is *permit*. This decision is sent to the PEP in the form of an XACML response as shown in Listing 6.2. The PEP will then permit *jsmith@users.example.com* to read the medical guide file.

Tr	e_1	e_2
Values	Subject= *jsmith@users.example.com* Action = *read* Object= *report − Bob*	Decision= *permit*

TABLE 6.2: Example of a Collected Trace from the PEP Request and Response.

The requests and responses respectively send and received by the PEP are collected in order to construct our trace. This trace is filtered to keep only those information that are needed for the testing process. In Table 6.2 we give an example of the trace built from the above example.

LISTING 6.3: Example of a Request.

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
http://docs.oasis-open.org/xacml/2.0/
access_control-xacml-2.0-context-schema-os.xsd">
<Subject>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>urn:example:role-values:physician</AttributeValue>
</Attribute>
</Subject>
<Resource>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:resource:view"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>urn:example:view-values:medicalReport</AttributeValue>
</Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:action:activity"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>urn:example:activity-values:read</AttributeValue>
</Attribute>
</Action>
<Environment>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:environment:context"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>urn:example:environment-values:designatedDoctor
</AttributeValue>
</Attribute>
</Environment>
</Request>
```

The collected trace can actually verify if the system give access to a user with a role physician to read a medical guide file. However, it does not provide any indication about the context in which this user has this privilege. According to the security rule this user can read a medical report only in the context *designated doctor*.

We show that existing XACML profile is extended with a context handler entity to manage abstract entities (that are role, view and activity) and the context. The value of the context is specified in the environment attribute of the request sent from the context handler to the PDP. This information is required in order to verify a security rule. Listing 6.3 shows the request which is sent from the context handler to the PDP. This request corresponds to the example given in Listing 6.1. The context value in this request is specified in the environment attribute as *designatedDoctor*.

In some cases giving the adequate rights for a user to perform an activity on resources may be constrained by obligations. Within XACML, the concept of obligations rules can be used. An obligation is a directive from the Policy Decision Point to the Policy Enforcement Point on what must be carried out after an access is granted. Obligations rules are returned to the PEP for enforcement. If the PEP is unabled to comply with the directive, the rights for a role to perform an activity may or must not be given. These obligation rules are specified in the response message of the PDP. The data about the obligation rules to be enforced by the PEP are also collected in our trace. In Listing 6.4, we provide an example of an obligation to notify a doctor after editing one of his/her patient's medical reports.

Listing 6.4: Example of an Obligation.

```
<Obligations>
<Obligation
ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
FulfillOn="Permit">
<AttributeAssignment
AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
DataType="http://www.w3.org/2001/XMLSchema#string">
$\&$lt;AttributeSelector RequestContextPath=
"//md:/record/md:patient/md:doctorInCharge/md:email"
DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt ;
</AttributeAssignment>
<AttributeAssignment
AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
DataType="http://www.w3.org/2001/XMLSchema#string">
Your medical report has been accessed by:
</AttributeAssignment>
<AttributeAssignment
AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
DataType="http://www.w3.org/2001/XMLSchema#string">
$\&$lt;SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt;
</AttributeAssignment>
</Obligation>
</Obligations>
```

Let us note that a collected trace based on the passive testing architecture contains information of the requests and the responses generated and received by the PDP respectively and also the request generated by the context handler. Therefore, this trace contains information about the subject, the action requested by this subject, the object on which the subject performs the activity and the access decision of the PDP. In the case where the right for the role to perform an activity on the view is constrained by obligations, the information related to these obligations are also collected. An example of a collected trace is presented in Table 6.3.

Tr	e_1	e_2
Values	Context= *designatedDoctor* Subject= *jsmith@users.example.com* Action = *read* Object= *report − Bob*	Decision= *permit* Obligation= NULL

TABLE 6.3: Example of a Collected Trace.

This trace has to to is the permission for a physician to read a medical file in the context designated doctor. Therefore this trace should be verified by the two LTL formulae that describe this security rule. The LTL formulae that describe this permission rule are as follows.

$$F7: \ designatedDoctor \rightarrow (jsmith@users.example.com \wedge read \wedge report - Bob \rightarrow \neg(deny \vee indeterminate) \cup permit)$$

$$F8: \ \neg designatedDoctor \rightarrow (jsmith@users.example.com \wedge read \wedge report - Bob \rightarrow \neg permit \cup (deny \vee indeterminate))$$

When applying our passive test algorithm by giving this trace and the two LTL formulae as an input the algorithm output is PASS. This means that the trace respect the permission rule.

6.5.7.4 Experimental Results

The test generated on the step 1 are applied to a prototype of our system under test. We performed our test to check the 53 security objectives that were defined in the step 1 of our testing process. The trace was collected according to our passive testing architecture. After applying our passive testing algorithm on the collected trace, the result of this test was a PASS, which means that according to the information contained in the trace, all the security rules were respected. Note that in order to obtain the final verdict the tester performs a simple boolean operation that combines the verdict of each LTL formula. If any of the verdict is FAIL the final verdict is FAIL.

6.6 Conclusions

In this chapter we presented the limitations of the active and passive methodologies for checking the interoperability security policies. In order to test the interoperability security policies, we proposed to combine these two approaches. An interoperability test case generation technique is adapted for testing interoperability security policies. The generated test cases from this step are used to stimulate the system that represents the O-grantor organization. The passive tester collects the executions of the O-grantor system to build an event trace. The trace is analysed in order to verify whether the behavior of the O-grantor respects its interoperability security policies. In addition, our method can be easily adapted to check other security policies relating information such as subject to role assignment, action to activity assignment and object to view assignment. This information can be verified based on static assignment specified in the security policy specification or dynamic assignment such as trust negotiation [HCBCD09]. In the latter, the properties to be verified has to be dynamically adapted with the new requirement based on the negotiation.

"A wise man will make more opportunities than he finds."
Francis Bacon

Chapter 7

Conclusion

Contents

7.1 Our Proposal

In this Thesis, we proposed two frameworks for testing and verifying the behavior of respectively a systems or a model with respect to its interoperability security policies. Our approaches take into consideration the interoperability security policies exigency such as the dynamicity, the diversity in the security rules modalities and the distributed nature of the system under test. Such security policies are in charge of administrating the interaction between collaborating systems. Thus in our study, we see our system under test as a distributed system that consist of at least two systems.

Our test framework consists of combining both the active and the passive testing. We show that these approaches are complementary when they are applied to interoperability security policies. Indeed when applying the active testing, the first step is to be able to model the interoperability security policies using formal models. In this Thesis, first we present a method to integrate the interoperability security policy in the function model of a system. In our integration process we consider that an interoperability security rule can have two different types of contexts: a state based and an event based context. The former consists of a logical condition that has to be satisfied in the system in order to activate the security rule (such as the location of the user) whereas the latter describes an event that has to be performed in the system. In addition, we consider three different types of security rules modalities such as permission, prohibition and obligation. Finally by using our integration method, security policies that are integrated in a functional model can be

easily updated. This feature is required when we consider dynamic creation or destruction of security rules based on the interoperability environment. Additional constitution of this Thesis is that we presented a framework for verifying the behavior of systems models with respect to its interoperability security policies. Our approach is a model checking based approach and so we first describe the interoperability security rules with a formal language. In this study, we show how security rules specified in O2O can be automatically described using the LTL. We described an O2O security rule with two LTL formulae. Moreover, we provided a mapping between the elements of the security rules and the functional model. Furthermore, it is possible to directly instantiate a security rule described in LTL with the values that correspond in the functional model.

It is another motivation that in this thesis we have provided a framework for testing interoperability security policies. In this step we show that active and passive testing are complementary for testing interoperability security policies. Then we detailed the process of active and passive testing steps of this framework.

Finally, in order to illustrate this appears, a hospital network case study is carried out throughout this Thesis to show the applicability of the methodology. A hospital network or health care system consist of two or more hospitals owned, sponsored, or contract managed by a central organization. Example of such hospital system can be: Adventist Health International, Aga Khan Health Services, Apollo Hospital and etc.

7.2 Discussion

7.2.1 Modeling Security Policies

In this Thesis we proposed a method to integrate interoperability security policies in the functional model of a system. This method aims at restraining the behavior of a functional model according to a security policy. Thus, the behavior of this model will respect these security policies. In some cases the behavior related to some security rule (i.e. obligation rules) does not exist in the initial functional model. In such cases, this behavior has to be properly described and integrated in this model. To integrate a security rule, we need first to locate the transitions in which this security rule has to be integrated. Thus we provided a mapping between the elements of a security rule and the elements that correspond in the functional model. Note that the automatization of this step is not simple. It is not always straightforward to find the elements in the functional model that corresponds to the elements of a security policy. For instance, in our approach an action of a security policy is related to a request (an input message) in the functional model. However if there is not a relation that directly link this action the the input message of the functional model, the mapping has to be done manually by constructing a table that defines the correspondence between these elements.

Another possible way is to model the functional behavior and the security policies in two different models that interact with each other. The interaction between these two model will describe the behavior of the system that respects these security policies. In this case, the initial functional model of the system has to be properly extended to include

an additional behavior that allows for this model to communicate with its security policy model. Moreover, this modeling approach leads to a massive interaction between these two models.

7.2.2 Interoperability Security Policies Verification

In order to verify the correctness of a model with respect to security requirements, we proposed a model checking based verification method. Security rules are described in LTL formalism. This formalism is compatible with the SPIN model checker. The main advantages behind using model checking are manifold:

- It is completely automatic and fast. In addition, there exist free and efficient tools such as SPIN, SMV, Murphi, or Uppaal for performing model checking based verification.
- Model checking can be used to check partial specifications, and thus it can provide useful information about system correctness even if the system is not completely specified.
- It produces counter examples which represents subtle error in design, and thus it can be useful for detecting these errors.

The main drawback of model checking is the state space explosion problem that happens when performing exhaustive state space search of some large systems. If such problem happens, another verification technique has to be adopted such as Theorem Proving [EFN71]. Existing work in this domain such as [YAB+05, BM11, BOKB10, UC06] have to be investigated and properly adapted to our context. Let us remark that, theorem proving is, in general, harder than model checking. It requires considerable technical expertise and understanding of the specification. Another disadvantage of using theorem proving is that if you fail to complete the proof of a property, then the tool will not tell you whether the property is indeed false or whether the user is not providing enough information to complete the proof.

7.2.3 Active versus Passive Testing for Interoperability Security Policies

Both active and passive testing techniques can be applied for testing interoperability security policies. In the case of active testing, the systems that are involved by the interoperability security policy have to be modeled using formal models. Then the interoperability security policy has to be integrated in these models. The strength of the active testing is in the ability to focus on particular aspects of the implementation, i.e. security requirements. In Chapter 6, we presented two possible test architectures and the test case generation process. In addition we discussed the limitation of each architecture when performing active testing. On the other hand, passive testing is defined as an online testing approach. Traces are collected from the interaction between these systems in run time. The passive testing will check whether the collected trace conforms with respect to the interoperability security policies. Note that using passive testing we can guarantee that the collected trace conforms to the interoperability security policy. Moreover, it is possible that some security requirements related to the security policy will not be tested. In such case the necessary

information that are related to these requirements are not included in the trace. In order to test these security requirements a longer trace has to be collected. Otherwise, the test verdict will be inconclusive.

7.2.4 Testing of Interoperability Security Policies

Each of the passive and active testing techniques has their limitations when they are used to check on interoperability security policies. However, as we showed in chapter 6 that these two approaches are complementary when used together. Using active testing technique, we generated test cases that are useful for stimulating the system under test. The main objective of these test cases is to build a trace that contains all these needed behaviors to test the interoperability security policy. This trace is collected through probes that are installed in some strategic points in the implementation. Note that by testing this trace we test whether these behaviors respect the interoperability security policy. Thus, we overcome the problem of an inconclusive verdict that could be generated when the trace does not contain all the needed information for the testing process.

7.3 Perspectives

At the end of this Thesis, we believe that several issues deserve to be addressed. Indeed, several issues identified during the development of this Thesis, are yet not treated.

First, in this thesis we based on the O2O model to specify interoperability security policies. Therefore, an organization (O-grantor) that wants to share resources with other ones has to define and implement an interoperability security policy for each organization that it wants to interoperate with. For this reason, in our testing process we proposed a test architecture that is constituted of only two systems: a system representing an O-grantee organization and another one representing an O-grantor. However, one can consider that each organization has a global interoperability security policy that administrates the access from several other organizations. In such case, the interoperability security policy of an O-grantor involves several O-grantees. Note that testing the interoperability security policy by applying the test process between the O-grantor and a randomly chosen O-grantee is not sufficient. This interoperability security policy can be different regarding each O-grantee organization. Contexts can be used to activate the security rules depending on the organization that generated the access request. Thus, some security rules can be active for an organization but not for another. Accordingly, we may need to include in the test architecture all the systems that are involved by the interoperability security policy.

Second, in our study we consider that the access right of a user does not automatically change in time, only the activation of an access rule is dynamic depending on some contexts. Studies such as access right based on trust evaluation are not discussed in this thesis. An organization can dynamically (in run time) evaluate the trust level of a user of another organization in order to define his/her access right. In such case, testing only the security policy is not sufficient anymore. The trust evaluation process and the access right assignment for a user have to be coupled with the security policies in the testing process

in order to properly address this issue. Otherwise we cannot guarantee the global security of a system.

Third, in multi organizations environment it is very frequent that organization propose services such as delegation. In the field of access control, delegation is an important aspect that is considered as a part of administration mechanisms. For instance, [BGTCCBB10] presents a flexible approach for extended role-based access control models to manage delegation and revocation. This approach provides means to express various delegation and revocation dimensions in a simple manner. Such delegation and revocation has to be studied in both the modeling and the testing levels.

Finally, there is many existing work in the domain of verification of security policies properties such as consistency, completeness and termination. However, most of these works does not consider dynamic security policies. In this case, the contexts of the security policies have to be considered in the verification process.

Bibliography

[ACNn03] José Antonio Arnedo, Ana Cavalli, and Manuel Núñez. Fast testing of critical properties through passive testing. In *Proceedings of the 15th IFIP international conference on Testing of communicating systems*, TestCom'03, pages 295–310, Berlin, Heidelberg, 2003. Springer-Verlag.

[AHCBCD06] Diala Abi Haidar, Nora Cuppens-Boulahia, Frederic Cuppens, and Herve Debar. An extended rbac profile of xacml. In *Proceedings of the 3rd ACM workshop on Secure web services*, SWS '06, pages 13–22, New York, NY, USA, 2006. ACM.

[Bar] Steve Barker. Information security: A logic based approach. In *International Conference on Enterprise Information Systems,*.

[BBF01] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. In *ACM Transactions on Information and System Security (TISSEC)*, volume 4, pages 191–233, New York, NY, USA, August 2001. ACM.

[BCDP05] Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. Geo-rbac: a spatially aware rbac. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 29–37, New York, NY, USA, 2005. ACM.

[BFS05] A. Belinfante, L. Frantzen, and C. Schallhart. Tools for test case generation. In *Model-based Testing of Reactive Systems: Advanced Lectures*, 2005.

[BGM02] Marius Bozga, Susanne Graf, and Laurent Mounier. If-2.0: A validation environment for component-based real-time systems. In *In Proceedings of Conference on Computer Aided Verification, CAV02*, pages 343–348. Springer, 2002.

[BGO+04] M. Bozga, S. Graf, I. Ober, I. Ober, and I. Sifakis. The if toolset. In *Lecture Notes in computer Science*, volume 3185, pages 237–267. Springer, 2004.

[BGTCCBB10] Meriam Ben-Ghorbel-Talbi, Frédéric Cuppens, Nora Cuppens-Boulahia, and Adel Bouhoula. A delegation model for extended rbac. *International Journal of Information Security*, 9(3):209–236, 2010.

[BK82] J. A. Bergstra and J. W. Klop. Fixed point semantics in process algebra. In *Technical report IW 280, Mathematical Centre*, 1982.

[BM04] P. Brezillon and G.K. Mostefaoui. Context-based security policies: a new modeling approach. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops 2004*, pages 154 – 158, march 2004.

[BM11] J. Bau and J.C. Mitchell. Security modeling and analysis. *Security Privacy, IEEE*, 9(3):18 –25, may-june 2011.

[BOKB10] Christoph Brandt, Jens Otten, Christoph Kreitz, and Wolfgang Bibel. Verification, induction termination analysis. chapter Specifying and verifying organizational security properties in first-order logic, pages 38–53. Springer-Verlag, Berlin, Heidelberg, 2010.

[Bos95] Anthony Boswell. Specification and validation of a security policy model. In *IEEE Transactions on Software Engineering and Methodology*, volume 21, pages 63–68, Piscataway, NJ, USA, February 1995. IEEE Press.

[BvS01] Martin Botha and Rossouw von Solms. The utilization of trend analysis in the effective monitoring of information security. part 1: the concept. volume 9, pages 237–242, 2001.

[BYNc07] Wan-Seob Byoun, Cheol-Jung Yoo, Hye-Min Noh, and Ok-Bae chang. Test case generation techniques for interoperability test of component based software from state transition model. In *Internation Journal of Computer Science and Network Security*, volume 7, pages 151–157, May 2007.

[Cad05] Tom Caddy. Penetration testing. In *Encyclopedia of Cryptography and Security*, 2005.

[CBCBCC08] Celine Coma-Brebel, Nora Cuppens-Boulahia, Frederic Cuppens, and Ana Rosa Cavalli. Interoperability using O2O contract. In *Fourth international conference on signal-image technology and Internet-based systems*, 2008.

[CC97] L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 103–, Washington, DC, USA, 1997. IEEE Computer Society.

[CCB08] Frederic Cuppens and Nora Cuppens-Boulahia. Modeling contextual security policies. In *International Journal of Information Security*, volume 7, pages 285–305, July 2008.

[CCBC06] Frederic Cuppens, Nora Cuppens-Boulahia, and Celine Coma. MotOrBAC : un outil d'administration et de simulation de politiques de sécurité. In *Security in Network Architectures (SAR) and Security of Information Systems (SSI), First Joint Conference*, June 6-9 2006.

[CCBCB06] Frederic Cuppens, Nora Cuppens-Boulahia, and Celine Coma-Brebel. O2O : Managing security policy interoperability with virtual private organizations. In *13th annual workshop of HP Openview University Association, HP-OVUA, may 21-24, Côte d'Azur, France*, 2006.

[CCBS05] Frederic Cuppens, Nora Cuppens-Boulahia, and Thierry Sans. Nomad: A security model with non atomic actions and deadlines. In *Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 186–196, Washington, DC, USA, 2005. IEEE Computer Society.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights*, volume 131, 1981.

[CLRZ99] Ana R. Cavalli, David Lee, Christian Rinderknecht, and Fatiha Zaidi. Hit-or-jump: An algorithm for embedded testing with applications to in services. In *International Conference on Formal Techniques for Networked and Distributed Systems*, volume 156, pages 41–56, 1999.

[COML08] Ana Rosa Cavalli, Edgardo Montes De Oca, Wissam Mallouli, and Mounir Lallali. Two complementary tools for the formal testing of distributed systems with time constraints. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 315–318, Washington, DC, USA, 2008. IEEE Computer Society.

[CS96] Fang Chen and Ravi S. Sandhu. Constraints for role-based access control. In *Proceedings of the first ACM Workshop on Role-based access control*, RBAC '95, New York, NY, USA, 1996. ACM.

[DBTS04] Michael Drouineaud, Maksym Bortin, Paolo Torrini, and Karsten Sohr. A first step towards formal verification of security policy properties for rbac. In *Proceedings of the Quality Software, Fourth International Conference*, QSIC '04, pages 60–67, 2004.

[Den87] D.E. Denning. An intrusion-detection model. In *IEEE Transactions on Software Engineering*, volume SE-13, 1987.

[Des11] Yves Deswarte. Protecting critical infrastructures while preserving each organization's autonomy. In *Proceedings of the 7th international conference on Distributed computing and internet technology*, ICDCIT'11, pages 15–34, Berlin, Heidelberg, 2011. Springer-Verlag.

[DFG+06] Vianney Darmaillacq, Jean-Claude Fernandez, Roland Groz, Laurent Mounier, and Jean-Luc Richier. Test generation for network security rules. In *18th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems (TestCom 2006)*, volume 3964 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2006.

[DV09] Alexandra Desmoulin and Cesar Viho. Formalizing interoperability for test case generation purpose. In *International Journal on Software Tools for Technology Transfer*, volume 11, pages 261–267, Berlin, Heidelberg, June 2009. Springer-Verlag.

[EFN71] Richard E., Fikes, and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Artificial Intelligence*, volume 2, pages 189 – 208, 1971.

[EKI10] A.A. El Kalam and N. Idboufker. Specification and verification of security properties of e-contracts. In *8th International Conference on Communications (COMM)*, pages 427 – 430, 2010.

[EMAHC09] Mazen EL Maarabani, Asma Adala, Iksoon Hwang, and Ana Cavalli. Interoperability testing of presence service on ims platform. In *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, pages 1 –6, april 2009.

[EMCA12] Mazen EL Maarabani, Ana Cavalli, and César Andrés. Testing interoperability security policies. In *The 24th International Conference on Software Engineering and Knowledge Engineering*, 2012.

[EMCHZ11] Mazen El Maarabani, Ana Cavalli, Iksoon Hwang, and Fatiha Zaïdi. Verification of interoperability security policies by model checking. In *HASE*, pages 376–381, 2011.

[EMHC10] Mazen El Maarabani, Iksoon Hwang, and Ana Cavalli. A formal approach for interoperability testing of security rules. In *6th international conference on Signal-Image Technology and Internet-Based Systems,SITIS2010*, 2010.

[ETS07] Methods for testing and specification (mts); internet protocol testing (ipt); generic approach to interoperability testing. 2007.

[FAl09] C. FAlm. An extensible framework for specifying and reasoning about complex role-based access control models. In *Technical Report MIP-0902, Department of Informatics and Mathematics, University of Passau*, 2009.

[FJFD01] Amgad Fayad, Sushil Jajodia, Donald B. Faatz, and Vinti Doshi. Going beyond mac and dac using mobile policies. In *Proceedings of the IFIP TC11 Sixteenth Annual Working Conference on Information Security: Trusted Information: The New Decade Challenge*, IFIP/Sec '01, pages 245–260, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[GBvS07] Robert Goss, Martin Botha, and Rossouw von Solms. Utilizing fuzzy logic and neural networks for effective, preventative intrusion detection in a wireless environment. In *Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 29–35, 2007.

[GK05] R. Grigore and H. Klaus. Rewriting-based techniques for runtime verification. In *Automated Software Engineering*, volume 12, pages 151–197, Hingham, MA, USA, April 2005. Kluwer Academic Publishers.

[Hai08] Diala Abi Haidar. *Web services access negociation.* PhD thesis, RSM - Dépt. Réseaux, Sécurité et Multimédia (Institut Mines-Télécom-Télécom Bretagne-UEB), 2008.

[HBM98] R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed environment. In *IEEE Symposium on Security and Privacy*, volume 0, Los Alamitos, CA, USA, 1998. IEEE Computer Society.

[HCBCD09] Diala Abi Haidar, Nora Cuppens-Boulahia, Frederic Cuppens, and Herve Debar. Xena: an access negotiation framework using xacml. In *Annals of telecommunications*, volume 64, pages 155 – 169, January 2009.

[HGXA06] Baik Hoh, Marco Gruteser, Hui Xiong, and Ansaf Alrabady. Enhancing security and privacy in traffic-monitoring systems. In *IEEE Pervasive Computing*, volume 5, pages 38–46, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[HK10] Hejiao Huang and Helene Kirchner. Formal specification and verification of modular security policy based on colored petri nets. In *IEEE Transactions on Dependable and Secure Computing*, volume 99, 2010.

[HLSG04] Ruibing Hao, David Lee, Rakesh K. Sinha, and Nancy Griffeth. Integrated system interoperability testing with applications to voip. In *IEEE/ACM Trans. Netw.*, volume 12, pages 823–836, October 2004.

[HMM+00] Amir Herzberg, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 2–, Washington, DC, USA, 2000. IEEE Computer Society.

[Hol03] Gerard Holzmann. *Spin model checker, the primer and reference manual.* Addison-Wesley Professional, first edition, 2003.

[HXCL12] J.H. Hwang, T. Xie, F. Chen, and A.X. Liu. Systematic structural testing of firewall policies. In *IEEE Transactions on Network and Service Management*, volume 9, pages 1 –11, march 2012.

[JBLG05] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, pages 4–23, Piscataway, NJ, USA, January 2005. IEEE Educational Activities Department.

[JJ05] Claude Jard and Thierry Jeron. Tgv: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. In *The International Journal on Software Tools for Technology Transfer*, volume 7, pages 297–315, Berlin, Heidelberg, August 2005. Springer-Verlag.

[KBM+03] Anas Abou El Kalam, Salem Benferhat, Alexandre Miege, Rania El Baida, Frederic Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 120–, Washington, DC, USA, 2003. IEEE Computer Society.

[Lam71] Butler W. Lampson. Protection, 1971.

[LBN99] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 291–298, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[LDB+93] G. Luo, R. Dssouli, G.V. Bochmann, P. Venkataram, and A. Ghedamsi. Test generation for the distributed test architecture. In *Networks, 1993. International Conference on Information Engineering '93. 'Communications and Networks for the Year 2000', Proceedings of IEEE Singapore International Conference on*, volume 2, pages 670 –674 vol.2, sep 1993.

[MBCBN08] W. Mallouli, F. Bessayah, A. Cavalli, and A. Ben-Nameur. Security rules specification and analysis based on passive testing. In *Global Communication Conference GLOBCOM*, 2008.

[MOC+07] Wissam Mallouli, Jean-Marie Orset, Ana Cavalli, Nora Cuppens, and Frederic Cuppens. A formal approach for testing security rules. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 127–132, New York, NY, USA, 2007. ACM.

[MTCB05] M.C. Mont, R. Thyne, K. Chan, and P. Bramhall. Extending hp identity management solutions to enforce privacy policies and obligations for regulatory compliance by enterprises. In *In 12th HP OpenView University Association Workshop*, July 10-13 2005.

[MWCM10] W. Mallouli, B. Wehbi, A. Cavalli, and S. Maag. Formal supervision of mobile ad hoc networks for security flaws detection. In *Book chapter in*

"Security Engineering Techniques and Solutions for Information Systems: Management and Implementation", 2010.

[OC06] Jean-Marie Orset and Ana Cavalli. A security model for olsr manet protocol. In *Proceedings of the 7th International Conference on Mobile Data Management*, MDM '06, pages 122–, Washington, DC, USA, 2006. IEEE Computer Society.

[pat] Intrusion detection system. In *Artificial Intelligence*.

[Roe99] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA99*, November 1999.

[RZFG99] Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies with complex constraints. In *Proceedings of the Network and Distributed System Security Symposium*, pages 89–107, 1999.

[SBC05] Diana Senn, David A. Basin, and Germano Caronni. Firewall conformance testing. In *Int. Conference on Testing of Communicating Systems (TestCom)*, pages 226–241, 2005.

[SDA05] Karsten Sohr, Michael Drouineaud, and Gail-Joon Ahn. Formal specification of role-based security policies for clinical information systems. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 332–339, 2005.

[SDAG08] Karsten Sohr, Michael Drouineaud, Gail-Joon Ahn, and Martin Gogolla. Analyzing and managing role-based access control policies. In *IEEE Transactions on Knowledge and Data Engineering*, volume 20, pages 924–939, July 2008.

[SKCK04a] Soonuk Seol, Myungchul Kim, Samuel T. Chanson, and Sungwon Kang. Fsm based interoperability testing methods for multi stimuli model. In *Roland Groz and Robert M. Hierons, editors, TestCom, volume 2978 of Lecture Notes in Computer Science*, pages 60–75, 2004.

[SKCK04b] Soonuk Seol, Myungchul Kim, Samuel T. Chanson, and Sungwon Kang. Interoperability test generation and minimization for communication protocols based on the multiple stimuli principle. In *IEEE Journal on Selected Areas in Communications*, volume 22, pages 2062–2074, 2004.

[SL06] Guoqiang Shu and David Lee. Message confidentiality testing of security protocols - passive monitoring and active checking. In *International Conference on Testing of Communicating Systems (TestCom)*, pages 357–372, 2006.

[Slo94] Morris Sloman. Policy driven management for distributed systems. In *Journal of Network and Systems Management*, volume 2, pages 333–360, 1994.

[TCEM12] Khalifa Toumi, Ana Cavalli, and Mazen El Maarabani. Role based interoperability security policies in collaborative systems. In *International Symposium on Security in Collaboration Technologies and Systems*, 2012.

[TG11] T. Tuglular and G. Gercek. Mutation-based evaluation of weighted test case selection for firewall testing. In *Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, pages 157 –164, june 2011.

[Tre02] Jan Tretmans. Testing techniques. In *Lecture notes, University of Twente, The Netherlands*, 2002.

[UC06] D. Unal and M.U. Caglayan. Theorem proving for modeling and conflict checking of authorization policies. In *International Symposium on Computer Networks*, pages 146 –151, 2006.

[XAC11] extensible access control markup language (xacml) version 2. In *Standard, OASIS*, February, 2011.

[YAB+05] Paul Youn, Ben Adida, Mike Bond, Jolyon Clulow, Jonathan Herzog, Amerson Lin, Ronald L. Rivest, and Ross Anderson. Robbing the bank with a theorem prover. Technical Report UCAM-CL-TR-644, University of Cambridge, Computer Laboratory, August 2005.

[ZHL06] Zhao-Li Zhang, Fan Hong, and Jun-Guo Liao. Modeling chinese wall policy using colored petri nets. In *Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, CIT '06, pages 162– , Washington, DC, USA, 2006. IEEE Computer Society.