MODEL DRIVEN CERTIFICATION OF CLOUD SERVICE SECURITY BASED ON CONTINUOUS MONITORING

Maria Krotsiani

City University London

Department of Computer Science

School of Mathematics, Computer Science and Engineering

Supervisor: Professor George Spanoudakis

Thesis submitted for the degree of Doctor of Philosophy,

January 2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis is a result of a long process, through which I have encountered difficult situations and some weaknesses of mine. Thus, I would like to thank all the people that supported me throughout this period.

Firstly, I want to express my sincere gratitude and appreciation to my supervisor, Professor George Spanoudakis, for his continuous encouragement, support and assistance. He was always patient, encouraging and really helpful by assisting me through many brainstorming meeting to solve any issues or difficulties I have encountered. His way of analyzing and expanding thoughts and ideas was quite painful sometimes, but it was really useful for my work. Without his guidance and continuous support throughout this period, I would not have been able to finish this thesis. I am really grateful that I had Professor George Spanoudakis supervising my research.

I would also like to express my sincere appreciation to my second supervisor, Dr. Christos Kloukinas, for all his insightful comments, his valuable advices, his constant support and his relaxing way to discuss any issues or difficulties I came across.

Furthermore, I would like to express my sincere thanks to Dr. Evangelia Kalyvianaki for her valuable support, encouragement and concern, especially in the last phase of the thesis preparation. I would also wish to thank Dr. Khaled Mahbub and Dr. Luca Pino for their valuable help, especially during the implementation phase of this thesis.

In addition to the above people, I would like to thank my collegues and friends that I made over these years, for their support and for the great time we had together. These people include Spyros Katopodis, Icamaan da Silva, Clara Rubio, Roberta Iacovelli, Dr. Ricardo Contreras and Dr. Rafael Roque Aschoff. Moreover, I would like also to thank all partners of Cumulus Porject for our great collaboration, and all members of the Department of Computer Science and the Technical Support Team (TST) of City University, for their continuous support.

My deepest gratitude goes also to my closest friends for their encouragement, understanding and patience. So, thank you Anastasia Michali, Evagelia Vasilarou, Dr. Aspasia Simpsi, Christos Paschos and Liana Kypriotaki.

Besides all the mentioned people, I would like to express my special thanks and gratitude to my parents, Eleftheria and Viktor, and my sister Elena, who were always there for me, especially during my breakdowns. Thank you for your support and love.

Last but not least, I would like to thank Sotirios for his constant support, help and advice through all these years. He was always there for me, in every difficult situation I have encountered, by showing understanding and by giving me valuable advices. Without his trust on my skills and his insistence to push me over my limits, I would have never started this PhD. Thank you for believing in me.

DECLARATION

The author grants powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to her. This permission covers only single copies make for study purposes, subject to normal conditions of acknowledgement.

ABSTRACT

Cloud Computing technology offers an advanced approach for the provision of infrastructure, platform and software services without the need of extensive cost of owning, operating or maintaining the computational infrastructures required. However, despite being cost effective, this technology has raised concerns regarding the security, privacy and compliance of data or services offered through cloud systems. This is mainly due to the lack of transparency of services to the consumers, or due to the fact that service providers are unwilling to take full responsibility for the security of services that they offer through cloud systems, and accept liability for security breaches [18]. In such circumstances, there is a trust deficiency that needs to be addressed.

The potential of certification as a means of addressing the lack of trust regarding the security of different types of services, including the cloud, has been widely recognised [149]. However, the recognition of this potential has not led to a wide adoption, as it was expected. The reason could be that certification has traditionally been carried out through standards and certification schemes (e.g., ISO27001 [149], ISO27002 [149] and Common Criteria [65]), which involve predominantly manual systems for security auditing, testing and inspection processes. Such processes tend to be lengthy and have a significant financial cost, which often prevents small technology vendors from adopting it [87].

In this thesis, we present an automated approach for cloud service certification, where the evidence is gathered through continuous monitoring. This approach can be used to: (a) define and execute automatically certification models, to continuously acquire and analyse evidence regarding the provision of services on cloud infrastructures through continuous monitoring; (b) use this evidence to assess whether the provision is compliant with required security properties; and (c) generate and manage digital certificates to confirm the compliance of services with specific security properties.

Chapter One

INTRODUCTION

1.1 OVERVIEW

This thesis presents a framework that was developed in order to automate the process of the certification of security properties of cloud services, based on evidence acquired through continuous monitoring of such services whilst they are in operation. The thesis describes the research foundations and the architecture that was followed in order to develop the proposed framework and the way that was implemented. It also provides an evaluation of the proposed security certification framework based on experimental results, static analysis of properties of the framework and an evaluation based on views of experts in the field of certification.

The remaining of this chapter provides an overview of the research problem that this thesis covers, as well as the overall aim of the research and the objectives that were followed to achieve its aim. Finally, the chapter closes by summarising the main contributions of the thesis and providing an outline of the remaining chapters of it.

1.2 MOTIVATION AND RESEARCH CHALLENGES

Cloud computing is one of the latest technological trend of recent years, which changed drastically the way IT services are delivered. Cloud computing provides the possibility to utilise computational capabilities to offer data, storage, compute and software services, upon demand, without owning the computational resources used to offer them [174]. The use of cloud services has been spreading fast over the last few years, formulating a market that is expected to reach a value of $4.13bn by 2017 [183]. Despite its fast spread, however, the use of cloud services still raises significant concerns that prevent users to fully adopt them. These concerns are mostly

related to the security of cloud services, including breaches of integrity, confidentiality [44][117][133] and privacy of customer data on clouds [61][44][117]; spamming, wrapping and cross-site scripting attacks [61]; various forms of Denial-of-Service (DoS) attacks resulting in reduced application and data availability [112][44][125]; and Authentication, Authorization and Accounting (AAA) vulnerabilities of cloud services [44][117]. Thus, the use of cloud computing has generated research regarding the security of the data and services that it offers [112][61][44][80][117].

Researchers have made a significant progress delivering methods and tools for access control and identity management on cloud systems [10], secure storage protocols (e.g., proof-of-retrievability [41], proof-of-storage protocols [129]), encryption and key management [150], and secure virtualisation [156]. However, despite of this work, the security of cloud services is still not totally guaranteed. This is due to several vulnerabilities of cloud service provision that are related to the possibility of breaches of integrity, confidentiality and privacy due to multi-tenancy of services; to the interference between security mechanisms operating at different layers of cloud services (infrastructure, platform and software services); to the interference between security and cloud virtualization or optimisation mechanisms (e.g., spreading of DoS attacks due to load balancing in cloud federations [125]); and due to challenging administrative tasks of the cloud infrastructure (i.e. maintenance purposes).

Furthermore, the risk arising from these vulnerabilities is increased since there are dependencies between services at all layers of the cloud stack, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These layers may also evolve and interact dynamically, such as when changing the Virtual Machines (VMs), or configurations of platform services, or deployed software services. These dependencies and their dynamic changes make it difficult to introduce an appropriate pre-deployment and operation controls for assessing and guaranteeing the security [44][125], as well as they will require a more dynamic assessment that could cover changes occurred after the deployment. Moreover, the exact provision of cloud services is often obscure, making it difficult to assess cloud security through audit mechanisms.

Under these circumstances, increasing the trustworthiness of cloud services requires a continuous and transparent assessment of the security of these services. The existing certification mechanisms, which have been used for this purpose, originate from techniques developed to certify properties of generic software systems and IT infrastructures as for example techniques based on the Common Criteria model [118]. Such techniques focus mainly on systems with a static structure that operate under stable operational conditions and, therefore, they are not suitable for the cloud systems and services, which can change, dynamically. There has also been research focusing on the certification of service-based systems, whose structure can change dynamically [12]. However, even the latter research overlooks the deployment infrastructure and platform services that are involved in the provision of software services in a cloud and special cloud conditions that introduce vulnerabilities such as service co-tenancy. Moreover, SOA certification research [12][13] focuses on the use (as opposed to the production) of certificates and on certificates produced through pre-deployment testing and/or formal analysis of software services, without taking into account real and continuous cloud service monitoring data.

To address the limitations of current research with regards to cloud service security certification, this thesis proposes a novel cloud service certification approach. The proposed approach automates the assessment of the security properties of cloud services and the creation of digital certificates for security properties of cloud services, based on continuous monitoring of cloud services. More specifically, the evidence required for assessing and verifying security properties is acquired through continuous monitoring of cloud service operations. Hence, the evidential basis for the assessment of properties is able to cover contextual conditions that might not be possible to predict, test or simulate through other forms of assessment, which take part before deploying a cloud service, such as testing or static analysis.

Continuous monitoring is able to capture contextual conditions in cloud service provision, such as changes in the population of co-tenant services, the deployed virtualization and optimisation strategies and mechanisms in the cloud and/or network and middleware configurations in a cloud, which are difficult to take into account in static forms of assessment. The proposed framework is a “model driven” framework that enables the automation of the certification process of cloud services, through the definition of certification models. These models drive fully the execution of the certification process, as they define the assessment rules of

a specific security property that needs to be assessed, as well as the conditions and the life cycle of the generated certificates.

1.3 OVERALL RESEARCH AIM AND OBJECTIVES

The overall research aim of the thesis is to address the limitations of the current certification mechanisms, by:

To develop an integrated framework in order to support a continuous and automated certification process for security properties of cloud services (infrastructure (IaaS), platform (PaaS) and software application layer services (SaaS)). In the framework that this thesis proposed, the evidence required for assessing security properties of cloud services, will be acquired through continuous monitoring of cloud service operations.

In order to achieve the overall aim of this thesis, the following key objectives were pursued:

Objective 1:

Review the literature on cloud security certification and identify gaps and controls relevant to the overall research aim of the thesis.

This initial objective provides an analysis of the current situation in cloud computing area, with regards to the security of cloud service provisioning. Moreover it will present related research and practices that has been already done regarding cloud security and the current certification and audit approaches for cloud services. These topics define the subject area of this research and its terminology. The analysis of the related work should describe the different existing frameworks and highlight the strengths and weaknesses of each approach, in order to identify the gap that this research intends to fill.

Objective 2:

Investigate different forms of monitoring based certification and develop a language for defining models to specify how this type of certification should be realised in order to enable the automation of the certification process.

To address the complexity of interactions of cloud services, we have to define and develop a model that will be able to certify security properties, by continuously collecting evidence based on operational monitoring. Monitoring based certification models address the limitations of current certification processes, which are a way to define all conditions for the certification process, and the inability to provide continuous certification of cloud services in order to be able to identify any security breach that might occur immediately and take the proper actions concerning the issued certificate. Certification based on continuous service monitoring would increase awareness of the operational context of the certification process that is difficult to achieve with static certification approaches, such as testing [75].

Consequently, a first step was to define a language for expressing the monitoring rules for a specific security property of a service, and a model to define all relevant information regarding the whole certification process. Moreover, a certificate life cycle model was also defined to address all possible states that a certificate might take, which is based on the continuous monitoring of cloud service, thus all states and their transactions are defined, as well as the conditions that need to be met for every transaction or the actions that should trigger.

Objective 3:

Develop a certification infrastructure for automatically generating certificates, and for providing access to, and managing certificates, driven by monitoring certification models.

To realise monitoring based certification models identified in Objective 2, we have developed and integrated mechanisms and tools to support:

- The operational monitoring of services in cloud systems;

- The analysis of monitoring data gathered, in order to gather the evidential basis required for issuing the monitoring-based certificates of different security properties of cloud services and automatically generate them;
- A language to express the conditions that should be met in order to issue, suspend or revoke a certificate according to the needs of the authority that handles the certificate;
- The transparency of the evidence and the information about the certificate; and
- The storage, retrieval and management of certificates, to be able to make them available to cloud providers and cloud customers.

Objective 4:

Evaluation of the proposed certification framework, through certification scenarios based on and/or inspired from real systems.

To ensure its technical trustworthiness and industrial applicability, our proposed framework was evaluated according to some use case scenarios, which cover technical and operational aspects of the overall certification approach. Moreover, the language used to express the life cycle of the certificate was also evaluated about its accuracy through a model checking process. Furthermore, an expert survey-based evaluation concerning the way certification models should be defined was also conducted, as well as an experimental certification process, in order to evaluate the correctness of the monitoring process.

1.4 RESEARCH ASSUMPTIONS

In order to conduct this research and develop the proposed approach, the following assumptions were made, which were used as the starting points and directions for the work:

- The framework for the automated generation of monitoring-based certificate described as objective 3, makes usage of a monitoring tool called EVEREST to support the continuous monitoring process of cloud services' operations. This monitoring tool was implemented by the Software Engineering Group of City University (a more detailed description of the tool is given in Chapter 3).

- The received events from the operations of the service to be certified should have a correct format with all relevant information included to them, in order for the used monitoring tool to read them and to proceed with the monitoring process. For instance, each event should have a specific XML format with predefined tabs and operation names, so that the used monitoring tool can extract the necessary information to proceed with the monitoring process.

- The proposed monitoring-based certification process allows the certification of cloud services at run time, and is envisioned as a continuous process based on the monitoring evidence collected from the monitoring tool, and based a Certification Model provided by a user, as an XML file.

- The focus of the research is on being able to correctly define assessment rules and conditions for a security property using an XML- based language in the Certification Model, and being able to process all this information in conjunction with the monitoring results, in order to certify a cloud service.

- Finally, all involved parties in the certification process are assumed to be trusted parties. This means that they should comply with a set of security dispositions to assure the security of the process.

1.5 RESEARCH CONTRIBUTIONS

This research aims to develop a novel model driven approach for continuous certification of cloud service security. The proposed framework provides a monitoring infrastructure that is be able to continuously monitor security properties of cloud services, tailored to the need of the relevant user, based on a customisable certification model. Based on the users' certification model, it can then process the information and proceed with the generation of the certificates for the specific cloud service and security property defined.

The main contributions of the research described in this thesis include:

- *Development of an XML based language for expressing monitoring based certification models*

  This contribution aims to provide a novel monitoring based certification process by introducing certification models for continuous assessment of security properties of cloud services. In order to achieve this goal, a new XML based language was defined for the specification of certification models, which enables the automated realization of certification models for an objective assessment of security properties, using evidence gathered through continuous monitoring.

- *Definition of an XML based Assertion Language to express the assertions of the security property*

  The new language that we have introduced, allows the definition of security properties' assertion rules and assumptions, in a way that can be understood by the monitor, in order to proceed with the monitoring of the service's operations and to produce the relevant monitoring results.

- *Development of a framework to process the Certification Models and automatically generate monitoring-based certificates.*

  This framework allows the processing of the defined certification models and the generation, or changing of status, of monitoring-based certificates. More specifically,

it is able to translate the security property assertions, defined in the certification model, to the EC-Assertion language that the monitor understands, and to process all conditions defined in the certification model, as well as the monitoring results received by the monitor, in order to decide and change the status of the certificate based on this information.

- *Evaluation of the approach*

  This contribution aims to prove the accuracy, the correctness and the performance of the proposed approach. Different evaluation methods were used in order to prove: i) the correctness and accuracy of the new language for defining monitoring rules, ii) that the monitoring process does not cause much overhead for gathering monitoring results, iii) that statically verifiable models can be defined and checked before they are put in operation, and iv) the experts' opinions about defining the proposed certification model and about the monitoring based certification process.

1.6 PUBLICATIONS

The contributions of this thesis have been also submitted and published to different conferences. Bellow all published papers are presented.

- Krotsiani M., Spanoudakis G., Mahbub K. "Incremental Certification of Cloud Services", In Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013), Barcelona, pp. 72-80, 2013 [145].

  In this paper, an early version of the certification model and the monitoring-based certification process was presented and described.

- Krotsiani M. and Spanoudakis G. "Continuous Certification of Non-Repudiation in Cloud Storage Services", In Proceedings of the 13th IEEE International Conference on

Trust, Security and Privacy in Computing and Communications (TrustCom), Beijing, pp. 921-928, September 2014 [143].

In this paper we described an early version of the certification model and the certificates' life cycle, as well as we explained the certification process based on continuous monitoring. Moreover, we have introduced the definition of the Non-Repudiation security property in the monitoring language and we introduced the notion of anomaly in the certification model. We explained possible anomalies that might occur while certifying the Non-Repudiation security property and the way that they can be defined in the proposed certification model.

- Hudic A., Krotsiani M., Tauber M., Spanoudakis G., Lorünser T., Mauthe A. and Weippl E. R. R. "A Multi-Layer and Multi-Tenant Cloud Assurance Evaluation Methodology", In Proceedings of the 6th IEEE International Conference on Cloud Computing, Technology and Science (CloudCom 2014), Singapore, pp. 386-393, 2014 [120].

  This paper presents a methodology based on Common Criteria, for aggregating information regarding security properties of individual components of cloud applications from different layers of a cloud stack, in order to provide an overall and continuous assurance level evaluation of cloud services.. Our main contribution in this work was to develop a broader cloud assurance methodology based on the certification process proposed in this thesis.

- Krotsiani M., Spanoudakis G. and Kloukinas C. "Monitoring-Based Certification of Cloud Service Security", In proceedings of the On the Move to Meaningful Internet Systems: OTM 2015 Conferences, Vol. 9415, pp. 644-659, Springer International Publishing, October 2015 [144].

In this paper an advanced version of the proposed approach was presented, incorporating an elaborated scheme for: (i) assessing the sufficiency of evidence for producing certificates, (ii) defining a life cycle model, and (iii) executing certification processes according to precisely defined models of them. Moreover, the experimental evaluation of the monitoring-based certification process was also introduced and analysed, where the results concerning the performance of the proposed approach was presented, based on a Protection Profile of Common Criteria and on a case study involving the certification of the MySQL database server.

1.7 OUTLINE OF THE THESIS

This thesis is organised into eight chapters including this chapter. The rest of the thesis is structured as followed.

Chapter 2 provides the literature review. The chapter investigates technological frameworks, mechanisms (controls) and standards for achieving security in the cloud and for certifying cloud security. A critical review of these is provided and gaps are identified. It also introduces a conceptual framework for monitoring based certification, and the basic concepts used for the proposed certification process are defined.

Chapter 3 focuses on the technical background of our research, by describing techniques and frameworks used for the realisation of the proposed approach. More specifically, it presents the monitoring framework that our approach is based on, as well as the EC-Assertion language that it uses. Moreover, the framework used for the evaluation of the proposed framework is also presented and described.

Chapter 4 presents a high level overview of the proposed certification process based on continuous monitoring, which is the focus of this thesis. Moreover, it provides a detailed description of the Certification Model that was developed. More specifically, the definition and the use of each element and sub-element of the certification model are explained in details.

In Chapter 5 three examples of the Certification Model are provided, which was explained in Chapter 4. In particular, the first example provides the certification model for the Non-Repudiation security property and explains the way to define each element. The other two examples refer to two case studies, each of which was used as an example to present the definition of different security properties that were certified.

Chapter 6 provides the implementation of the proposed research, by presenting the architecture of the certification infrastructure, used for producing monitoring-based certificates, and by explaining its components and their methods. Moreover, three algorithms that were developed to support the monitoring based certification process are also presented and explained. These algorithms refer to i) the translation of the new Assertion language used in the certification model into the EC-Assertion language that the monitor uses, ii) the handling of the defined expected behavioural model, which is part of the proposed certification model, and iii) the handling of the life-cycle model defined in the certification model.

Chapter 7 presents the evaluation of the proposed framework and of the certification model. More specifically, a survey-based evaluation for the certification model was conducted to evaluate the correctness and accuracy of the XML based language used for defining the certification models. Furthermore, a model checking of the life cycle model is presented, as well as the set up and the results of an experimental evaluation of the proposed framework that was implemented as a proof of concept for our approach. The experimental evaluation has been based on a Protection Profile of Common Criteria and a simulated case study.

Finally, Chapter 8 concludes with the summary of the proposed approach that is described in this thesis and the contributions of the research underpinning it. It also describes the limitations of the approach, and outlines directions for future work on security certification.

Chapter Two

LITERATURE REVIEW

2.1 OVERVIEW

This chapter provides an overview of the current state of cloud computing in terms of security and privacy. Its purpose is to give readers the necessary technical background on cloud security and cloud security certification.

More specifically, section 2.2 covers the technical background on the concept of cloud security. It also provides a conceptual model, in order to bond all terms of cloud security, in order to give a better understanding of their associations. Section 2.3 covers the broad area of cloud security and its relevant elements. Concepts such as threats, vulnerabilities, and security risks faced in cloud computing, as well as security mechanisms and security properties are being analysed and discussed. Section 2.4 presents the concept of certification and certificates, in order to enable readers to understand the relationship of the current research to existing approaches of cloud certification. Finally, section 2.5 summarises the open issues faced in the current state of cloud certification, which are the motivations for the current research.

2.2 CERTIFICATION CONCEPT AND CONCEPTUAL MODEL

Cloud computing became a mainstream solution of Information Technology (IT), where resources and services are provided on demand and on a pay-as-you-go basis [21][22]. Infrastructure, platform and software services, known as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), respectively, are provisioned to clients, who can use these resources or services at lower prices and high performance and flexibility. Cloud Computing has transformed business and government by transforming the way

of designing and deliver technology and application, into a more efficient way. However, these advantages have created new security challenges by introducing new security vulnerabilities, including emerging security issues. These issues are the main reason that discourages cloud users from adopting these solutions [18].

In recent years, the security research community has worked hard to improve the security of cloud systems and several solutions have been developed. According to the current research [18][149][180], checking security properties is greatly facilitated whilst an accredited authority produces, maintains and manages signed certificates through the certification process. As *Certification Process* we can define an activity that takes as inputs the assertions for specific security properties of a specific cloud service that needs to be certified, in order to produce as an output a certificate, which will contain all relevant evidence that proves the requested properties, by means of verification and validation techniques. These certificates are produced concerning security properties of cloud entities, as well as the evidence supporting such assertions. Security properties refer to threats that might occur, which are generated by different threat agents, and they introduce the notion of risk. To mitigate these risks, different security mechanisms are being designed.

In order to provide a common understanding about the key concepts of cloud security, such as certification, certificates, security properties, assertion, threats, and risk; we provide a conceptual model that accumulates all these concepts. This model provides a basic conceptualisation of the proposed research and related notions, as well as their association. It is an extended version of the meta-model produced by the University of Milan for the Cumulus Project [69], which integrates also the Threat-Risk-Asset model introduced in Common Criteria [65], as well as the terms of *Security Compromise*, *Security Controls* and *Mitigation*.

A shown in Figure 1, there is a *Threat Agent* entity, who is responsible for generating and producing *Threats,* which refer to a *Security Property*. Consequently, a *Threat* introduces the *Risk* in cloud solutions, which causes a *Security Compromise* for the *Asset*. A *Security Compromise* is mitigated by the *Security Controls* developed in the *Asset*, which are also known as Countermeasures. The *Certification Model* assesses this *Mitigation*, in order to generate and issue a *Certificate* that certifies the *Target of Certification* for the specific *Security Property*.

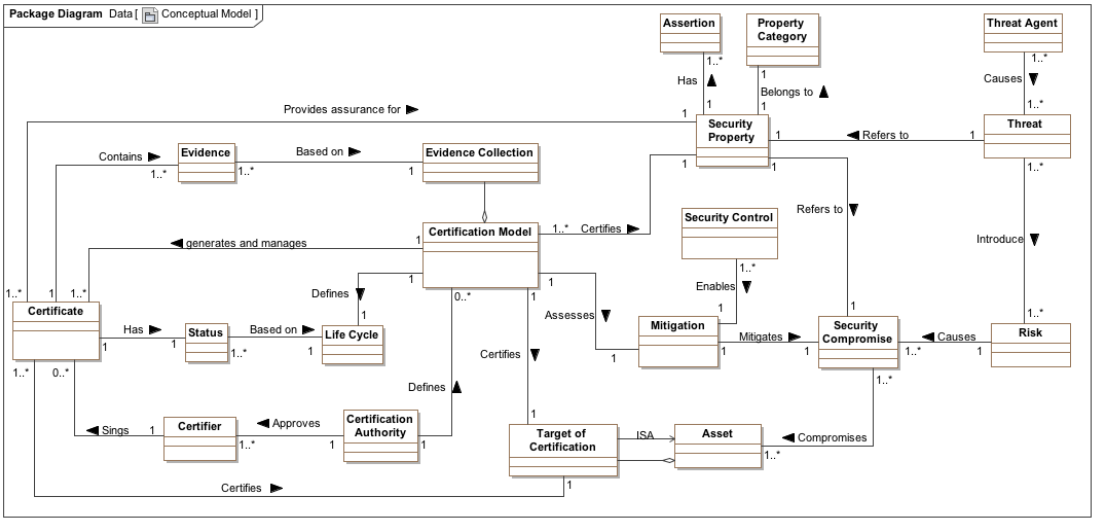


Figure 1 – Conceptual Model for Cloud Certification

Based on this model, the following definitions can be introduced for a better understanding of the terms used.

Definition 1:

A *Certification Model (CM)* is a model that defines the way in which it is possible to obtain assurance on whether a given security property is preserved by an asset, for a given period of time, generate a certificate to encode information that is necessary for this assurance, and manage this certificate as evidence regarding the preservation of the property or otherwise accrues.

A *certification model* defines:

- The *asset* that is the target of the certification process (i.e., the target of certification),
- The *security property* of the asset for which assurance is required,
- The *evidence* that needs to be considered for establishing the necessary assurance of the asset, and

- The process of generating and updating certificates to express the assurance regarding the security property for the asset (referred to as "life cycle" model in the following).

Definition 2:

Security Properties are part of the Certification Model entity in our conceptual model. As *Security Property*, we define the security requirement imposed on a cloud service, which is derived from applicable laws, policies, standards or regulations, that need to be addressed and its measurements (e.g. sample size, period), in order to obtain the necessary assurance level.

Definition 3:

The term *Assertion* refers to the formal specification of a security property that is expressed in a manner allowing the automated assessment of whether the security property that it refers to is satisfied.

Definition 4:

The *Target of certification (ToC)* is defined as an asset of a cloud service (e.g. a specific service operation, a set of service operations, data managed by the service) or an asset that is required or contributes to the realization of a cloud service (e.g., a virtual machine), which becomes the subject of certification.

Definition 5:

The term *Life Cycle* refers to a state chart diagram, used to define all possible states a certificate can take during its life. It also defines the transactions between different states based on specified conditions that need to take place.

Definition 6:

As a *Threat* we can define any possible event that can have a harmful impact to organizational operations (such as functions, image, or reputation), organizational assets, or individuals, through an information system via unauthorized access, destruction, disclosure, modification of information, or denial of service. A threat can also be a potential source to successfully exploit particular information system vulnerability, or compromising the integrity of software code.

Definition 7:

The term *Risk* can be defined as the probability of occurrence of an unwanted event and its negative consequences for the stakeholders [205]. More specifically, ISO 27005 defines risk as "the potential that a given threat will exploit vulnerabilities of an asset or group of assets and thereby cause harm to the organization," assessing it in terms the probability of an event to happen and its consequences [122].

Based on the conceptual model, a *CM* includes all conceptual entities involved in the certification process of cloud entities and defines how a certificate is produced, what is its content, and how it is managed. Apart from the *Security Property* entity, a CM also includes i) a *Certification Authority* entity that approves a *Certifier* to sign the generated *Certificate,* ii) an *Evidence Collection* entity, iii) a Target of *Certification (ToC)* entity, and iv) a *Life Cycle* entity. Therefore, a *certification model* should define the *security property* that needs to be certified regarding a specific target of certification (*ToC*) that this property applies to and the way evidence should be collected assessing a property. A *Security Property* belongs to a specific *Security Property Category* and has one or more Assessments, which define the rules based on which the validity of the security property is being checked.

According to a certification model that is defined (or endorsed) by a certification authority, *certificates* are being produced and signed by a *Certifier*. Thus, after having defined a *CM* and checked the evidence of a *ToC* for a specific *Security Property*, a *Certificate* is generated. Each

Certificate should have a reference to the relevant *CM* that was generated from, and provide the *Evidence* that where used in order to issue it. Moreover, the *ToC* that it certifies should also be specified, as well as its *Status,* which changed according to the *Life Cycle* defined in the *CM*. When the *evidence* gathered for a certificate is sufficient for verifying the security property related to it (as determined by the certification model), a *certificate* could be issued. However, even after they are issued, certificates can be updated subject to changes in the operational conditions of the cloud service, which should also be defined in the certification model. All these possible updates and other key changes in the *status* of a certificate are defined in the *life cycle* entity of the *Certification Model*.

2.3 SECURITY PROPERTIES, THREATS, RISKS AND SECURITY CONTROLS

The key factor to manage risks in cloud computing is to recognise and understand the nature of security threats that exist. Therefore, in order to facilitate risk-management decisions regarding cloud adoption, a deep understanding of new threats and vulnerabilities that cloud computing adds with respect to security issues should first be achieved, as well as the identification of cloud threats that cause these issues.

2.3.1 SECURITY PROPERTIES

In order to assure customers about the security level that is being provided by a service, a form of assurance should be given through a commitment made by the cloud providers to their customers. To achieve this assurance, the security mechanisms, which are used to protect the cloud services from possible attacks, should be assessed and verified. Thus, the term *security property* represents the security requirements imposed on cloud service that should be verified. These requirements define the set of attributes that specify the details of the definition of the security property that should be verified, as well as the method to measure it.

Security evaluation mechanisms such as Common Criteria's "security functional components" [65] and the control domains of ISO 27002 [122], have taken a more practical approach to define

a series of security domains. A clear categorisation of security properties based on a classification scheme was produces in the Cloud Control Matrix (CCM) [59] by Cloud Security Alliance's (CSA), that provides a good coverage and a clear taxonomy for security properties.

CCM is a control framework focuses specifically on cloud security. It lists a series of control objectives on the lower level, which is slightly different from providing security properties, and on a higher level, the control domains that are defined cover the key areas that are critical to cloud computing [59]. Moreover, the CCM is designed to map also other well-known control frameworks such as ISO 27001 [122], COBIT [62] or PCI-DSS [186], in order to cover all possible security risks.

Thus, according to the CCM and to the work done by CSA in the CUMULUS Project [68], the main security property categories and their corresponding abstract security properties are:

1. Application & Interface Security

This category refer to the security of applications and APIs used in cloud environments, and it consists of the following abstract security properties:

- ***Data Integrity***. This property checks that data has not been changed, destroyed, or lost by an unauthorized user or in accidental manner.
- ***Confidentiality.*** This property checks that data is not disclosed to system entities unless they have been authorized to access the data.
- ***Authenticity.*** This property checks that the asset is genuine and able to be verified and be trusted.
- ***Non-Repudiation.*** This property checks that an entity cannot deny having participated in a part or a whole of a data transaction, such as uploading or downloading data.

- ***Information flow control.*** This property checks the possibility to monitor data exchanges between entities, for the purpose of blocking or alerting about deviations from the expected behaviour.

- ***Auditability.*** This property provides chronological records of system activities in a sufficient way, in order to enable the examination of the sequence of activities of an operation, procedure.

2. Infrastructure & Virtualization Security

This category refers to risks that may occur concerning the coordination and maintenance of the different cloud infrastructures and virtual machines used in cloud computing. The main abstract security property of this category is:

- ***Tenant Isolation.*** This property refers to the ability of an asset to keep its tenants segregated from each other.

3. Interoperability & Portability

This category refers interoperability and portability of cloud services and its abstract security property is:

- ***Portability.*** This property checks if data are usable in different environments, or different cloud service providers.

4. Security Incident Management, E-Discovery & Cloud Forensics

This category refers to incident management of cloud services. It consists of the following abstract security properties:

- ***Incident management quality.*** Checks the ability of an entity to handle and report incidents in a proper way.

5. Identity & Access Management

This category refers the way users' identity, access and privacy are being maintained and managed. The relevant abstract security properties of this category are:

- ***Identity assurance.*** This property obtains assurance concerning the identity of users, through the provision and verification of their credentials.
- ***Credential security.*** This property refers to the secure storage of users' credentials, which should be protected against external or internal threats.
- ***Account control.*** This property refers to the ability af an asset to keep its users' accounts secure, to protect them from malicious threats.
- ***Access privacy.*** This property provides users with access to resources without having the ability to identify them.

6. Encryption & Key Management

This category refers to the encryption used by the cloud providers and the way they store and manage the different encryption keys. In this category the relevant abstract security property is:

- ***Key management.*** This property protects the confidentiality and integrity of cryptographic keys during their whole lifecycle.

7. Governance & Risk Management

This category refers to the way risks are being handled in a service, and the relevant abstract security property is:

- ***Risk control.*** This property checks the ability of a system to analyse and quantify external or internal risks.

8. Legal & Standards Compliance

This category refers to the legal compliance of cloud providers and their compliance to standards. This category consists of the following abstract security properties:

- ***Location control.*** This property checks the ability of a system to restrict geographically the location where data is stored and processed.
- ***Personal data privacy.*** This property checks that the disclosure of data related to a physical person is under the control of that person, in terms of recipients and purpose of processing.

9. Data Security & Information Lifecycle Management

This category refers to risks concerning data security and the way data is being handled. The following abstract security properties are relevant to this category:

- ***Data disposal.*** This property refers to the ability of an asset to effectively dispose of data and all its copies/backups.
- ***Data leakage control.*** This property refers to the ability of an asset to monitor breaches of data confidentiality.
- ***Durability.*** This property refers to the ability an asset to protect the integrity of stored information during its whole lifecycle.

10. Data centre Security

This category refers to threats and risks concerning the physical integrity of an asset. The relevant abstract security property of this category is:

- ***Physical Integrity.*** This property obtains assurance to a physical entity, not to be bypassed by an unauthorized user.

11. Business Continuity Management & Operational Resilience

This category deals with different types of risks regarding the operational aspect of an asset. The relevant abstract security properties of this category are:

- ***Availability.*** This property refers to the ability of a system or a system resource to be accessible, usable or operational upon demand, by an authorized user, according to performance specifications of the system.
- ***Recovery.*** This property refers the ability of an asset to return to a secure state, in case of a failure or a malicious attack.
- ***Resource control.*** This property refers to the ability of an asset to control how resources are being allocated.

12. Change Control & Configuration Management

This category refers to the way cloud provides handle configuration changes and security policies. It consists of the following abstract security properties:

- ***Compliance control.*** This property refers to the ability of an asset to enforce security policies.

Configuration change control. This property refers to the ability of an asset to monitor and report configuration changes to customers.

Table 1 summarises all security property categories and their relevant abstract security properties that were presented.

Table 1 - Security Property Categories and Abstract Security Properties

Security Properties	
Security Property Category	**Abstract Security Property**
Application & Interface Security	Data Integrity
	Confidentiality
	Authenticity
	Non-Repudiation
	Information flow control
	Auditability
Infrastructure & Virtualization Security	Tenant Isolation
Interoperability & Portability	Portability
Security Incident Management, E-Discovery & Cloud Forensics	Incident management quality
Identity & Access Management	Identity assurance
	Credential security
	Account control
	Access privacy
Encryption & Key Management	Key management
Governance & Risk Management	Risk control
Legal & Standards Compliance	Location control
	Personal data privacy
Data Security & Information Lifecycle Management	Data disposal
	Data leakage control
	Durability
Threat & Vulnerability Management	Vulnerability management quality
Data Centre Security	Physical Integrity
Business Continuity Management & Operational Resilience	Availability
	Recovery
	Resource control
Change Control & Configuration Management	Compliance control
	Configuration change control

2.3.2 ASSETS OR TARGET OF CERTIFICATION

According to definition 4 that we gave in Section 2.2, a Target of Certification is an asset of a cloud service, such as a specific service operation or a set of service operations, as well as data managed by the service, or even an asset that is required or contributes to the realization of a cloud service (such as a virtual machine), which becomes the subject of the certification process.

Based on the cloud computing reference architecture provided by NIST [154] as shown in Figure 2, an asset or a ToC can be any component of the service orchestration, in any of the three layers of a could service (SaaS, PaaS, IaaS), the resource control or the physical resource area, as well as any component of the Cloud Service Management. Moreover, assets can also be humans or organisations that could be related to a cloud component and contribute with the relevant ToC for the security realisation. However, these types of assets are not standalone ToCs in the current research, but they are taken under consideration in case they are associated with another type of ToC.

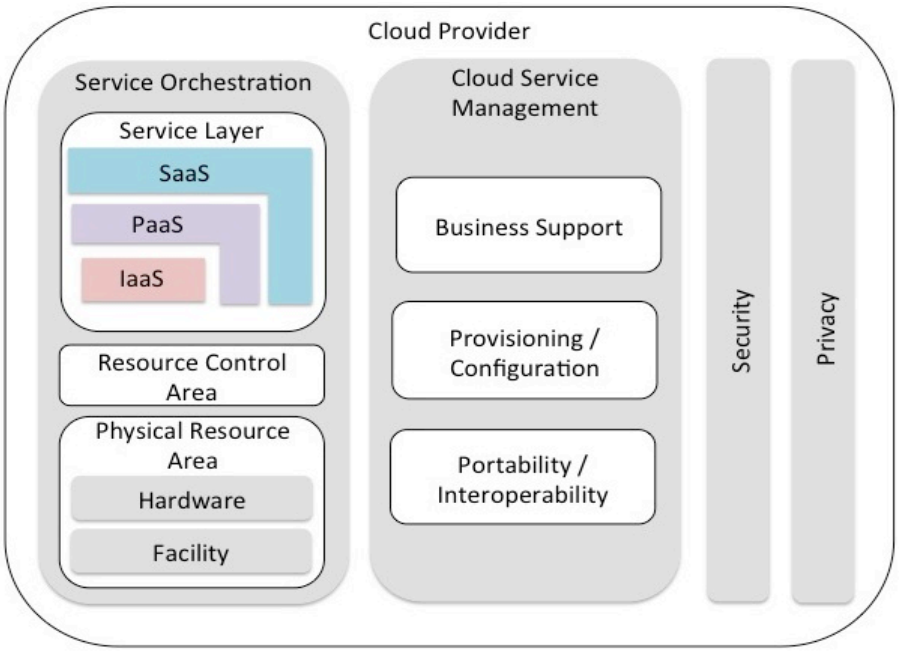


Figure 2 – Cloud Computing Reference Architecture

2.3.3 THREATS

As defined in Definition 6, a threat can have a malicious impact to organizational operations of an asset. Among the most significant security threats associated with cloud computing, is the tendency to bypass information technology (IT) departments and information officers [57]. According to the "Notorious Nine" report from Cloud Security Alliance (CSA) [57], the most critical threats identified in the area of cloud computing the last years, are:

1. ***Data Breaches***, which refer to incidents in which sensitive, protected or confidential data has potentially been viewed, stolen or used by an unauthorized individual.

2. ***Data Loss***, which refers to the permanently loss of data. This could be caused due to malicious attackers, an accidental deletion or even to a physical catastrophe.

3. ***Account Hijacking***, which refers to attacks such as phishing, fraud, or exploitation of software vulnerabilities to gain access to users' credentials and eavesdrop on activities and transactions, manipulate data, return falsified information, or redirect clients to illegitimate sites.

4. ***Insecure APIs.*** Cloud providers expose a set of software interfaces or APIs that customers use to manage and interact with cloud services. These APIs are used to perform provisioning, management, orchestration, and monitoring. Therefore, the security concerning authentication, access control, encryption and activity monitoring, as well as the availability of cloud services depend upon the security of these APIs, which must be designed to protect the services against both accidental and malicious attempts.

5. ***Denial of Service (DoS)***, which are designed to disrupt or disable services or resources, in order to prevent them from delivering traffic to legitimate users. They aim to make services and resources unavailable to its intended users, such as to temporarily or indefinitely interrupt or suspend cloud services or applications.

6. ***Malicious Insider***, which refers to the threat of a current or former employee of an organization, or any other business partner with authorized access to the organization's

network, system, or data, who intentionally exceeded or misused that access in a negative manner affecting the confidentiality, integrity, or availability of the organization's information or information systems.

7. ***Abuse of Cloud Services***, which refers to the mal-use of the vast amount of computing power provided by cloud systems. An example could be to use cloud-computing power to crack an encryption key faster.

8. ***Insufficient Due Diligence***, which refers to cases where some organisations rush to adopt cloud technologies without understanding the full scope of the undertaking that might lead to contractual issues over obligations on liability, response, or transparency by creating mismatched expectations between the cloud provider and the customer.

9. ***Shared Technology Issues***, which refer to threats caused due to the shared technology that cloud computing offers, such as the hypervisor, a shared platform component, or an application in a SaaS environment, which expose the entire environment to a potential of compromise and breach. This threat can affect an entire cloud at once.

Except of these nine threats, there are also some other identified threats known as ***Injection attacks***, such as SQL injection, command injection or cross-site scripting attacks. This type of attacks has the intention to inject malicious pieces of code or scripts to be executed in parts of the services or applications, in order to intercept data [211].

Molner and Schechter tried to classify various threats, which could not only be related to cyber-attacks, in *technical* and *non-technical* threats [175]. Even though technical threats, such as Denial of Service (DoS) attacks, are more critical, they suggest that non-technical threats are just as important as the technical ones, and they classify them into the following categories:

1. ***Contractual Threats***

   These threats are related to contractual issues when using cloud services, such as bankruptcy of stakeholders in a provisioning chain, potential switching costs between providers, or cost-overruns due to attacks that aim to maliciously consume resource,

such as the abuse of cloud services. This type of threats can cause reduced availability of cloud services.

2. ***Jurisdictional Threats***

   These threats can refer to either i) an indirect legal coercion, such as cases where cease and desist requests sent to the cloud infrastructure provider; or ii) to direct or indirect jurisdictional exposure from copyright holders and law enforcement agencies, such as exposure to "secret searches".

3. ***Organisational Threats***

   This type of threats can be challenging to assess, as they refer to threats that are caused by human interactions, which cannot be monitored and audited. An authority can check the application of organisational quality management standards, such as those related to the ISO 9000 standards, in order to determine if these aspects can be potentially spotted and addressed.

One of the main benefits that cloud computing offers is the shared tenancy of a physical data centre infrastructure, such as the ones in public or community clouds. However, shared tenancy introduces a number of potential new threats and risks, which are explored by Molner and Schechter [175]. According to the potential restrictions on forensic capabilities caused by malicious tenants, a threat that is difficult to monitor, audit, or has reduced response capabilities, may occur. Therefore, the shared tenancy that cloud systems offer can cause a non-technical threat that is related to the so-called jurisdictional collateral damage. This type of threats concerns situations where law enforcement agencies may request the shutdown of a data centre, due to a malicious behaviour of one of its tenants. Finally, specific technical threats from cloud tenants can be associated with direct breaches, side-channel attacks, denial of resources related to insecure APIs or resource theft.

In Table 2 below, all categories of the mentioned threats of cloud computing are summarized.

Table 2 - Threats Categories

Threats	
Technical	**Non -Technical**
Data Breaches	***Contractual***
Data Loss	Abuse of Cloud Services
Account Hijacking	***Organisational***
Insecure APIs	Insufficient Due Diligence
Denial of Service (DoS)	***Jurisdictional***
Shared Technology Issues	Malicious Insider
Injection attacks	

2.3.4 RISKS AND SECURITY CONTROLS

Risk may rise due to various factors such as the location of the data centres, data security and integrity, the lack of knowledge and/or misuse of the provided infrastructure, or due to the governing policies and regulations. In order to resist to an attack caused by a threat agent and decrease the impact or the probability of a threat to occur, strong security mechanisms are necessary to detect and avoid the attackers. Cloud computing service providers use various security measures to ensure that security risks are taken care of, in order to resist to various malicious attacks.

According to the literature, security risks and their security controls in cloud computing can be classified in different categories with regards to their objectives. The three main categories of security controls, based on the security properties they refer to, which are also known as *CIA*, are: 1) ***C****onfidentiality* [205][21][39][50][77][172][184]; 2) ***I****ntegrity* [205][21][184][234][94][107][206]; and 3) ***A****vailability* [39][172][184][43][61][133][204]. These main objectives are the most important for the risk assessment process, in order to assure cloud systems for adoption.

1. Confidentiality (C)

Confidentiality refers to the protection of personal and sensitive data or information, from unauthorized access or information disclosure. This includes means of maintaining personal privacy of users.

Confidentiality and strong authorization and authentication mechanisms are essential for cloud providers, in order to gain trust from their customers. Unauthorized access, theft of sensitive data or identification codes (credentials), and the increased number of credentials used by a user, which can cause the reuse of the same passwords in multiple services [172], can be identified as main attacks of this category. Consequently, as properties of interest for confidentiality, we can indicate the authorization of users [107][132][52], the security of users' credentials [234][110][132] and the restriction to unauthorised access [184][219][220][225]. Credential and identification mechanisms [172][234], authorization mechanisms [184][107][225], and cryptographic mechanisms [107][234] should be the targets, in order to evaluate the risk regarding this objective, and avoid the attacks related to confidentiality.

More specifically, these mechanisms can be further defined as:

a. Authentication Mechanisms

Authentication is the process of uniquely validating a particular entity or individual's credentials. In focuses in identifying whether a particular individual has the right to enter a system or access a secure site. Because the aim of authentication is to prevent unwanted access to resources, network's authentication credentials should be kept secure and secret to safeguard this information [224][52][53][184][110][132].

Most security policies state that to access a service, a user must enter a login ID and password that are authenticated by a security server [107][166][184][110][132]. To maximise security, one-time or dynamic can be used or strong password policies that define certain requirements that a password should meet (length, specific characters to be used or combinations of characters that are forbidden). Some times a password may be accomplished with the use of a token, which is a physical or virtual object that store authentication information, such as smartcards or ID badges [195]. Furthermore, biometrics mechanisms can also be used for authentication, which

authenticate users based on physical characteristics, such as fingerprint scanners, retina scanners or hand geometry scanner, as well as voice or facial recognition software [233][195]. Many systems might also use a multi-factor authentication mechanism, which requires at least two different authentication factors to be used, thus enhancing the authentication process [107][166][224][52][53][184][110][132].

b. Authorization Mechanisms

Authorization refers to the process that determines whether a user has the authority to perform various tasks. It grants users privileges by determining what type of activities, resources, or services they are permitted to use. Usually, authorization occurs within the context of authentication. Once a user is authenticated, they may be authorized to access or perform different types of activities [184][107][110][132][225][52].

2. Integrity (I)

Integrity refers to the protection against improper data modification or destruction from unauthorized users, in order to ensure the authenticity, validity, quality and security of data or information. Unauthorized changes and modifications in sensitive data or information [234] [94][107], loss of consistency due to massive data duplication [184][206], data loss or destruction and faulty clients that send inconsistent shares to different servers [21] are some of the threats of this risk category. As a result, the focus should be on data loss or data modifications [234][94][107], and data validity, quality, and durability of system's operations [184][234][220]. In order to resist to these threats, a system should have strong authorization [184][107][225], validity and quality mechanisms, which are important in Cloud computing, in order to ensure referential integrity is maintained [193].

3. Availability (A)

Availability refers to high-speed and reliable access, storage or use of any required service, information, or application, at any time and from anywhere users desire. The availability threats

could be the disruption of network connection or network failures [172], the disruption of data access or server failures [172], the overcapacity of the cloud [21][184][94][61][204], and the lack of data backups or insufficient data recovery services [21][133][202]. As a result, the main focus for the availability is on the reliable access, which should be at any time, the data access in general [21][220], as well as the overcapacity of the cloud. In order to resist to these threats, checking access mechanisms [43], cloud capacity measurements [184][94][61], priority of users mechanisms in case of overcapacity, as well as data recovery and backup services [172][133] should be properly evaluated.

Another relevant security mechanism concerning the availability risk is the *Quality of Service (QoS)*. QoS refers to the ability of a system to optimise performance, by prioritising certain network traffic, applications or data flows, to ensure a consistent level of performance. In other words, it measures the performance of a system, such as bandwidth, latency, error rate, uptime, delay or jitter. Due to the cloud characteristic of rapid elasticity [173][32], cloud services can migrate across different cloud infrastructures or they can scale up or down, which can compromise the QoS measures between components of a service, leading to low performance [209].

Except of these three main categories (CIA), three more categories can be defined to cover some other types of threats and risks in the cloud. These are:

4. Data Location (DL)

Data location refers to the aspect of cloud computing, in which service providers are not concentrated in a single location, but instead they are geographically distributed around the globe. This risk creates unawareness among cloud customers about the exact location of the data centres [94][117][49][138][119][202]. Moreover, but having distributed data centres users are not in control over the physical access mechanisms to their data stored in these cloud providers [202]. As a result, this could interrupt investigations within the cloud and makes it difficult to access some activities of the cloud, where data is not stored in a particular data centre but in a distributed format [49]. The location of a provider's data centres or operations can affect risk in different ways, due to jurisdictional threats, political instability, or actual geographical threats, such as the possibility of earthquakes in a region or the availability of reliable power sources [49][138][119].

In order to overcome this risk, according to [49][202], cloud providers should make a proper risk analysis and gain trust froth their users.

5. Data Segregation (DS)

Data in the cloud is typically in a shared environment together with data from other cloud users [49][138][119][202][166]. However, data isolation is not always easily achievable in all cloud environments, because not all data can be segregated according to the users' needs [117][49]. Encryption mechanisms are one effective way to achieve data segregation [234][94][107][57][117], but in some cases it can destroy completely the data, due to an unwanted accident [224]. A more secure way to use encryption mechanism is by providing some evidence concerning the encryption schemes that are being used by cloud providers.

6. Accountability (AC)

Accountability measures the resources that users consume. This can include the amount of system time or the amount of data a user has sent and/or received during a session. Accountability in cloud computing refers to a set of approaches in order to addresses two main problems: i) the lack of trust in cloud service providers and the difficulty they face to be compliant across geographic boundaries, and b) the emphasis on data protection [107][173][76][90].

Accountability can be achieved by logging sessions and usage information. This information is then audited and used for authorization control, billing, trend analysis, resource utilization, and capacity planning activities [107][173][76][90].

In the table below, we summarize the risks and their security mechanisms used to overcome them.

Table 3 - Cloud Risks and Security Mechanisms

Risks	Security Mechanisms
Confidentiality (C)	Authentication
	Authorisation
Integrity (I)	Authorisation
	Validity mechanisms
	Quality mechanisms
Availability (A)	Access control
	Capacity measurements
	Quality of Service
Data Location (DL)	Trust
Data Segregation (DS)	Encryption
Accountability (AC)	Audit and logging mechanisms

2.3.5 SUMMARY OF SECURITY PROPERTIES, THREATS, AND SECURITY CONTROLS

Table 4 gives a summary of the concepts that this section presented and maps them together, in order to provide a better understanding of their connection.

Table 4 - Combined Vulnerabilities, Threats and Security Risks in Clouds

Security Property Category	Abstract Security Property	Threats	Risks
Application & Interface Security	Data Integrity	Data Breaches, Shared Technology Issues, DoS	C, I, A, DS
	Confidentiality	Injection Attacks, Shared Technology Issues, Data Breaches, Malicious Insider	I, AC
	Authenticity	Data Breaches, Account Hijacking	C, I, A

	Non-Repudiation	Data Breaches, Malicious Insider	C, I, A
	Information flow control	Injection Attacks, Shared Technology Issues, Data Breaches	C, I, A
	Auditability	Injection Attacks, Shared Technology Issues, Data Breaches, Malicious Insider	C, I, A, AC
Infrastructure & Virtualization Security	Tenant Isolation	Data Breaches, Shared Technology Issues	C, I, A, DL, DS
Interoperability & Portability	Portability	Shared Technology Issues	C, I, A, DL
Security Incident Management, E-Discovery & Cloud Forensics	Incident management quality	Insufficient Due Diligence, DoS	A, AC
Identity & Access Management	Identity assurance	Account Hijacking, Abuse of Cloud Services, Injection Attacks	C, I, A
	Credential security	Account Hijacking	C, I, A
	Account control	Data Breaches	C, I, A
	Access privacy	Data Breaches Account Hijacking DoS	C, I, A
Encryption & Key Management	Key management	Data Breaches Account Hijacking DoS Abuse of Cloud Services	C, I, A, DS
Governance & Risk Management	Risk control	Abuse of Cloud Services	C, I, A
Legal & Standards Compliance	Location control	Data Breaches, Shared Technology Issues	C, I, A, AC, DL, DS
	Personal data privacy	Data Breaches, Account Hijacking, Injection Attacks, Malicious Insider	C, I, A
Data Security & Information Lifecycle Management	Data disposal	Data Breaches, Account Hijacking, DoS	C, I
	Data leakage control	Data Breaches, Account Hijacking, DoS	C, I

	Durability	Data Breaches, Account Hijacking, DoS	C, I
Threat & Vulnerability Management	Vulnerability management quality	Injection Attacks Shared Technology Issues, Data Breaches, Malicious Insider	I, AC
Business Continuity Management & Operational Resilience	Availability	Insufficient Due Diligence, DoS	A
	Resource control	Data Loss Insufficient Due Diligence	A, AC, DL
Change Control & Configuration Management	Compliance control	Insufficient Due Diligence Abuse of Cloud Services	C, I, A, AC
	Configuration change control	Insecure APIs Shared Technology Issues Injection Attacks	C, I, A, AC

2.4 CERTIFICATION AND CERTIFICATES

Certification of service providers is an essential tool to build trust between them and their customers. It is a type of industry agreed "compliance" mechanism, through which providers can be certified in order to demonstrate adherence to particular standards [149]. By using certified entities, customers can rely on the assessed security properties of an asset, provided that the certification process that is used for the assertion is able to produce sufficient evidence for the validity of the specific property.

Hence, certification allows users to formulate a trusted judgment for cloud providers, according to their compliance with certain policies or standards. The outcome of the certification process is usually a *certificate*, which states the cloud providers' compliance according to an assessment defined in a certification model. When a certificate claims adhering to certain certified policies, trust comes from compliance with those same certificate policies. In other words, a certificate indicates that an issuing certification authority (CA), by following specified policies, states that a cloud provider adheres to specified policies. An important aspect of certification is

the transparency of the data produced by the cloud infrastructure and the collected evidence to certify it, since it make cloud security issues more clear to end users [217].

There are many certification schemes, each of which provides different benefits and challenges. As mentioned in Section 2.2, a *Certification Model (CM)* is used as a guideline for the certification process and includes all conceptual entities involved in the certification process. Focusing on the type of the asset that needs to be certified, or as we mentioned it the *Target of Certification* (ToC), we can define three main categories: i) software certification, ii) service certification and iii) cloud certification. Moreover, concerning the way evidence is being collected, there are different ways to achieve it, such as: i) by self-assessment, which includes an investigation by a third-party authority, or by filling Questionnaires, ii) based on Trusted Platform Modules (TPM), iii) by conducting tests, or iv) by monitoring. The following sub-chapters describe these different types of the certification process and give relevant existing work done in this area.

2.4.1 CERTIFICATION METHODS BASED ON TOC

In this section we provide an overview of the existing research regarding certification processes based on the type of ToC that needs to be certified. Targets of certification can be either software, services or cloud services.

Software certification covers a wide range of formal, semi-formal, and informal evaluation techniques, including formal verification of compliance with explicit safety policies, system simulation, testing, code reviews and human “sign offs”, or even references to supporting literature. Subsequently, the certificates can be derived from different types of certification process based on the mechanism that is being used. However, current certification approaches are not yet suitable for cases where a high level of dynamism is required [15]. To be more precise, in their current form, they cannot be used to support and automate run-time security assessment.

Lately, certification process started being an important part in the Service-Oriented Architecture (SOA) environment, for certifying service functional and non-functional properties [15][72]. As a continuation of their evolution, certification schemes need to adapt to new

environment by incorporating the definition of solutions that fits also in the cloud environment, so as to provide trustworthiness in cloud-based services and applications. Certification of cloud-based services and applications are similar to SOA certification, as both infrastructures are highly dynamic. The difference between the two lays in some extra requirements of cloud systems, which introduce the need to manage and certify the whole cloud-computing stack, including IaaS, PaaS, SaaS.

2.4.1.1 SOFTWARE CERTIFICATION

Software certification has a long record in the certification research. Software certification has been initially used for assessing functional properties of software systems. More recently it has been mainly applied for certifying non-functional properties of software, including security.

The process of the security certification [73] is expensive in terms of time and effort. The first standard for software security certification was the Trusted Security Evaluation Criteria (TCSEC) standard by U.S. Department of Defence, commonly referred to as the Orange Book, which was developed in 1985 (USDoD 1985) [78]. The Orange book was used to design and develop security requirements, in order to address the communication gap between vendors, evaluators, and customers [153]. Later on in the Nineties, the demand of software security certification outside the U.S. lead to the development of other security certifications standards, such as Information Technology Security Evaluation Criteria (ITSEC) in Europe (ITSEC 1991)[123] and the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) in Canada (CSE 1993) [26]. These national certification schemes are totally independent from each other and as a result, the cost to certify a software system at an international level has remained very high for a long time. Thus, in order to fulfil the need of an affordable international software security certification standard, the Common Criteria (CC) certification standard has been defined. Common Criteria provides a unified process and a flexible framework, which can be used to specify, design, and evaluate security properties of different IT products [118]. The main goal of the CC evaluation is to certify that the security policies claimed by the developers are correctly implemented by the security functions. In CC, users can specify their security functional and assurance requirements, vendors can then implement and or make claims about the security attributes of their products,

and testing laboratories can evaluate the products to determine if they actually meet the vendors' claims. In other words, Common Criteria provides assurance that the process of specification, implementation, and evaluation of a computer security product has been conducted in a thorough and standard manner. An important aspect to consider, which will be also relevant in cloud-based certification, is the definition of a Certificates Management Scheme (CMS) to find a mean to extend the validity of a Common Criteria Certificate after the end of the certification process. The main reasons of loss of the validity of a Common Criteria (CC) certification are: i) the discovery of new vulnerabilities; ii) the inclusion of new functionalities into the target of evaluation; and iii) the relevant modifications to the security target (e.g. new security functionalities, changes in security problem definition).

Another research in this area is the ICSA security certification, which provides a less complex software certification process [121]. Its goal is to alleviate certification costs that previous approaches had, to certify software products across multiple versions and configurations, and to provide a simple, but thorough, security assurance process for network and Internet-related software products.

The Certification Commission for Healthcare Information Technology (CCHIT) software certification [46] is a domain-specific certification process, which aimed to provide liability reduction for healthcare-related software products. The Certification de Sécurité de Premier Niveau (CSPN) [7] is another solution for a lightweight security assessment proposed by the Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI).

Overall, the goal of software security certification process is to provide the possibility to an independent authority to determine the reliability and security of software systems, without the need of using the techniques and tools required for the certification process itself. However, these methods are more focused on systems with a stable structure that operate under stable operational conditions, rather than a more dynamic ones, like cloud services.

2.4.1.2 SERVICE CERTIFICATION

The work done in the field of security certification schemes have mainly focused on solid software components, as discussed in the previous section, which usually provide human-readable certificates used during the deployment and installation time. Consequently, the approaches proposed so far for software security certification could not support a service-based scenario, since it requires the availability of machine-readable certificates, and their integration within service selection and composition frameworks [72]. Hence, research started focusing also in the area of service certification processes.

On the contrary of software certifications mechanisms, the actual state of the art of service certification mechanisms do not include any standard yet, but the research still continues. The main challenges faced by service certification research are in a sense preliminary and envisage for the challenges that also the research in the field of cloud certification will face. Similarly to the software testing, the artefacts used for supporting service certification can be of two types, either model proofs or test proofs.

The modelling of systems in the context of web services, including their workflow and their business processes, demand specific treatment. An industry standard for specifying systems workflows is the Web Services Business Process Execution Language (WSBPEL) [162], or BPEL in short. Other industry driven modelling approaches [215][141] are based on the Unified Modelling Language (UML) [197]. For UML there are also some extensions that enable to a certain extend the modelling of security properties [128][155] to be provided by the system. One of the more recent approaches of this type is the AVISPA [25], which was funded by the European Union in the FET Open program. AVISPA provides the High Level Protocol Specification Language (HLPSL [51]) and four different analysis tools. One of these tools, the SAT–based Model Checker [19][66], was recently used to identify a security flaw in the SAML–based Single–Sign–On protocol for Google Applications [20]. Since AVISPA uses specification structures focusing on crypto protocols, this approach is currently being extended towards reasoning for dynamic composition of services within the EU funded project AVANTSSAR [24]. Another approach that has recently been extended has been conducted by Gürgens and Rudolph, which has been used to find security flaws in a number of key exchange, authentication and non–

repudiation protocols [109][108]. This approach is supported by the Simple Homomorphism Verification Tool [99] and has also been applied to analyse certain scenarios based on Trusted Computing. In the FP7 Project SERENITY [208] their approach was further extended to cover Ambient Intelligence scenarios that are characterized by dynamic context changes. Though the research in the field of certification has being adapted for the verification of service-based systems, using one of the above approaches in the certification process, the respective certificates will not contain any information regarding the verification outcome.

2.4.1.3 CLOUD CERTIFICATION

Since Cloud Computing has been introduces in the area of IT, the research on certification has started growing in this field, too. The research done on certification of cloud-based services and applications is still in an early stage. In the last years many cloud certification schemes have been defined, but there are still some unresolved issues, such as i) issues related to the adequacy and complexity of standards; ii) process and administration issues; iii) issues related to transparency and public communication; and iv) limitations in the assessment process [149].

In a cloud-computing environment, which is a dynamic environment, to reassure the security compliance is difficult. The cloud is opaque and the cloud providers can have different security mechanisms in place. Khan et al. [136] envision the usage of certification to fully materialize a trusted cloud model, which will be able to support security and trustworthiness for cloud-based services and applications. Heiser and Nicolett [117] have evaluated the cloud security risks and reached the conclusion that cloud-computing environments share IT risks with any externally provided service. In addition, they have also indicated some unique cloud attributes that require risk assessment in areas such as data integrity, recovery and privacy, and an evaluation of legal issues in areas such as e-discovery, regulatory compliance, and auditing. Finally, they have also predicted that the certification will become the norm for cloud offerings in the near future.

One requirement for the provision of sustainable Cloud Computing is the ability to certify the adherence of business processes to regulatory requirements. The business models behind service technologies like SOA and cloud computing essentially depend on business processes being

tailored to the individual needs of customers. Accorsi et al. [5] focus on providing business processes on cloud and preliminary face the problem of automated certification of cloud-based business processes using a Petri net approach. They state that a key obstacle to the development of large-scale, reliable cloud computing is the difficulty of timely compliance certification of business processes operating in rapidly changing cloud environments. Reliable cloud computing must provide control over business process compliance. In this context, the central task is certifying business processes for their adherence to regulations. The authors propose the usage of ComCert [64] as a method of automated compliance certification of business processes, and to apply it in the cloud environment. ComCert is a tool that auditors can used to check cloud-based processes for adherence to a set of different compliance requirements, in order to detect vulnerabilities arising from the control-flow and dataflow perspectives, such as whether all required activities are included or whether activities happen in the prescribed order. This approach is based on Petri nets, which are used to decide on the policy adherence of business processes, by providing an expressive, notation-independent formalism to capture the semantics of business processes [4]. ComCert can be used for auditability, accountability or portability security properties.

The Cloud Industry Forum (CIF) [55] provided another work on cloud certification in 2009, to provide transparency through the certification process and to assist end users in determining essential information about the provisioning of cloud services, in order to adopt them The CIF Code of Practice was revised and published in 2010. To claim compliance with the CIF's Code of Practice, cloud providers need to conduct an annual self-certification and confirm the successful results to the CIF, in order to receive the certification "mark". Optionally, an organisation may choose the independent certification performed by a CIF-approved certification body. Moreover, CIF will check and investigate randomly any formal complaint of non-compliance of a provider against the Code.

COBIT [62] is another approach, the purpose of which is to provide cloud providers with an information technology (IT) governance model, to facilitate the understanding and management of the risks associated with IT. It is a control model to ensure the integrity of information and information systems. COBIT provides globally accepted principles, practices, analytical tools and

models, in order to increase the trust of information systems. Its criteria are based on quality, trust and security requirements.

The Federal Risk and Authorization Management Program (FedRAMP) [92] is a government-wide program that provides a standardized approach for security assessment, authorization, and continuous monitoring for cloud products and services. It enables cloud providers to obtain a provisional authorization after undergoing a third-party security assessment. FedRAMP assessment process uses the FedRAMP requirements, which are compliant with FISMA [91] and are based on the NIST 800-53 rev3 [180]. In order undergo this assessment, cloud providers should implement the FedRAMP security requirements on their environment and an approved third party assessment organization will perform an independent assessment of the cloud system, in order to provide a security assessment package for review. Then, the FedRAMP Joint Authorization Board (JAB) will review the assessment package in order to grant a provisional authorization. Federal Information Security Management Act (FISMA) is a compliant framework that consists of a set of standards and quality assessment criteria, used to examine the information security planning conducted by federal government agencies [91].

McAfee Cloud Security Platform [169] is a certification framework that combines also an automated vulnerability audit. It is designed to provide security measures for clouds and to protect the communication between cloud user and cloud service providers. It includes audits of existing security controls and processes, as well as mitigation of vulnerabilities, and reporting of the security status of the service, by a respected third party. Rather than adopting the unique security practices and policies of each cloud vendor, McAfee Cloud Security allows businesses to extend and apply their own access and security policies, by securing all data traffic and data storage.

TRUSTe provides a commercial cloud certification service, called “TRUSTed Cloud Privacy Certification”, that aims to certify the privacy functionalities supported by service providers [228]. TRUSTed Cloud Privacy Certification is especially important for service providers that work in EU and are subject to European Union Safe Harbor Principles in the management of sensitive data. TRUSTe provides a means to certify compliance with the EU directive on data protection. It ensures practices, such as type of data collected, data flows, data security measures

and data usage policies that should meet the program requirement criteria, which are based on transparency and accountability.

Finally, a most recent certification framework was introduced by CSA for cloud services, called CSA Security, Trust and Assurance Registry (STAR) [60]. This is a research project that covers key principles of transparency, auditing, and synchronisation of standards. The CSA Open Certification Framework [182] will support several tiers, recognizing the varying assurance requirements and maturity levels of providers and consumers. These will range from three levels of assurance that the STAR provides from self-assessment to high-assurance specifications that are continuously monitored. The open certification framework is structured on the following three levels of trust:

LEVEL ONE: ***STAR Self-Assessment***. In this level cloud providers can either submit the Consensus Assessments Initiative Questionnaire (CAIQ), or submit a report to indicate their compliance with CSA Cloud Controls Matrix (CCM) [59].

LEVEL TWO: This level consists of two different types of certification based on the way evidence is being collected. These types are:

i) CSA STAR Attestation. This type of attestation uses the requirements of an AICPA (American Institute of Certified Public Accountants) SOC 2 attestation that is being conducted in accordance with AT section 101 of the AICPA attestation standards, combined with the CSA Cloud Controls Matrix (CCM) [59].

ii) CSA STAR Certification. The second type of this level is a third party assessment that is being conducted by using the requirements of the ISO/IEC 27001:2005 management systems standard [122] combined with the CCM [59].

LEVEL THREE: CSA STAR Continuous Monitoring. This level of assurance provides a monitoring-based certification, in order to automate the current security practices of cloud providers currently under development. This level is still under development.

Table 5 below summarises the different types of certification process and methods, based on the type of the Target of Certification (ToC).

Table 5 - Certification Processes based on ToC

Certification based on ToC	
Software	Orange Book [78]
	ITSEC [123]
	CTCPEC [26]
	CC [118]
	ICSA [121]
	CSPN[7]
Services	AVISPA [25]
	AVANTSSAR [24]
	Simple Homomorphism Verification Tool [99]
	SERENITY [208]
Cloud Services	ComCert [64]
	CIF Code of Practice [55]
	COBIT [62]
	FedRAMP [92]
	McAfee Cloud Security Platform [169]
	TRUSTed Cloud Certification Platform [228]
	CSA STAR [60]

2.4.2 CERTIFICATION METHODS BASED ON EVIDENCE COLLECTION

In this section we provide an overview of the existing research regarding the way evidence can be collected to verify the security property of cloud services for the cloud certification process. The type of evidence collection can be either i) based on assessments regarding specific standards or regulations, performed by either the cloud providers or third party authorities, known as self-assessments ii) based on trusted platform modules (TPM), ii) based on performing tests, or iii) based on continuous monitoring.

2.4.2.1 SELF- ASSESSMENT CERTIFICATION METHODS

The majority of the certification schemes for cloud services are based on self - assessment certification process, which is conducted either from the cloud provider itself or by hiring a third party accredited certification authority. According to this process, the cloud provider should provide compliance to specific standards or security requirements that the chosen certification scheme defines.

As we discussed in the different cloud certification schemes, in the cases where cloud providers conduct the assessment process by themselves, they should either complete a specific questionnaire provided by the scheme, such as in the case of CSA STAR Level 1 and Level 2 regarding the CSA STAR Attestation scheme [60], or by completing reports regarding specific national or international standards, such as the ComCert [64], the CIF Guidance [55], the COBIT [62], the compliant framework FISMA [91], and the TRUSTe [228].

Moreover, there are cases where certification schemes require the intervention of a third party accredited authority to provide the assessment, in order to certify the provided services, such as in the case of FedRAMP scheme [92], CSA STAR Level 2 STAR Certification [60] and McAfee Cloud Security Platform [169].

2.4.2.2 TPM BASED CERTIFICATION METHODS AND SECURITY CONTROLS

Trusted computing relies on collecting evidence based on TPMs [176] and on related hardware, which are used to prove the integrity of software, processes or data. Trusted Computing technologies are well suited to provide proofs of the trustworthiness on the lower level of the cloud stack, which is the hardware layer. A TPM has the capability of providing secure storage and basic cryptography primitives (including key generation/storage, encryption and digital signatures). The TPM is therefore a trusted element that is physically secure and can ensure the security of a platform [229].

Based on the TPM, the Trusted Computing Group defines a series of functionalities that can be used to increase the security of different systems. Among these, remote attestation is one of the core functionalities. The basic idea is to verify that a given platform is trustworthy (i.e. the platform is in a known and trusted state) before establishing a communication with it. To achieve this goal, the TPM provides dedicated registers called Platform Configuration Registers (PCR) to store a platform configuration in the form of a hash value. Depending on the scenario, PCRs can also be digitally signed [152]. Based on stored PCRs, the remote attestation mechanism allows changes to a user's computer configuration to be detected by authorized parties (validating parties) [63].

Krautheim [142] defines a private virtual infrastructure, which can be used to share the responsibility between users and cloud providers, in order to decrease the overall risk of exposure. This approach is based on the notion of Virtual Trusted Platform Module (vTPM), which was introduces by Berger et al. [34]. The vTPM provides storage and cryptographic functions of TPM for applications and operating systems that run in Virtual Machines (VMs). Boampong and Wahsheh [38] introduced a work stating that security in cloud systems could be achieved by enriching cloud with a trusted computing platform (TCP). Based on their work, Santos et al. [203] introduce the Excalibur. The Excalibur is a system that provides data confidentiality and integrity by encrypting data according to a customer-defined policy. In this way it guarantees that the encrypted data can only be decrypted when the configuration matches the used policy.

Cloud Trust Protocol (CTP) is the mechanism that allows users to request and retrieve information about the cloud provider infrastructure. According to Ardagna et al. [17], in order to provide assurance in cloud systems and to have balanced security processes and controls between providers and customers, both introspection and extrospeciton should be taken under consideration by cloud providers and cloud customers (tenants in general). Introspection is the capability of a cloud provider to examine and observe its internal processes, whereas extrospection is the ability of customers and service providers to examine and observe cloud's internal processes, involving their activities, data, and applications, for security purposes. Furthermore, transparency is an essential requirement to support both introspection and extrospection. Furthermore, Munoz and Mana [178] propose an approach for cloud certification that combines software and hardware-based certification. Their approach is based on trusted

computing technology and aims to bridge the gap between cloud certification and trusted computing.

Finally, Li et al. [151] present the MyCloud, which is an architecture used for privacy protection based on traditional encryption mechanisms. MyCloud decreases as much as possible the trusted computing base and the cloud providers' ability to modify privacy settings, in order to provide to the clients the ability to configure their privacy protection.

2.4.2.3 TESTING-BASED CERTIFICATION METHODS AND SECURITY CONTROLS

Concerning the test based artefacts, service testing differs from standard testing practices, as the insecurely nature of web services causes major restrictions to the way testers can interact with the services during the testing process. Thus, research has mostly focused on addressing the problem of testing for web services, by assessing the correctness of a service, and by automatically generating test cases for service verification [28][42]. King and Ganti [137] introduce the first solution for autonomic self-testing in clouds. They combine an automated test script for cloud services with a Test Support as-a-Service (TSaaS), in order to provide partial automated testing activities for remote cloud services. Tsai et al. [230] first propose an approach by using and extending WSDL standard to cope with web service testing, for service composition in clouds. Frantzen et al. [98] describe another approach for web service testing, which is based on the modelling of services as symbolic transition systems. The proposed solution aims to generate run-time tests suitable for testing the coordination of services. Keum et al. [135] proposed an approach in the same field, which was based on extended finite state machines, that enriches WSDL with information about the dynamic behaviour of services to automatically generate test cases and improve the testing coverage. Salva and Rabhi [201] describe an approach for testing the robustness of web services, by automatically producing test cases from the WSDL. The proposed approach verifies the correctness of each operation in the WSDL and their robustness using so called "hazard values". Finally, Zech [235] presents a model-driven methodology, in which tests are generated according to negative requirements, which are derived from the risk analysis done for a service. Some works have also focused on the definition of specification-based testing solutions for web services [115][127][164][116][81].

Damiani et al. [73] first study the problem of assessing and certifying the correct functioning of SOA using security certificates based on signed test cases. A first step in the certification of SOA and Web Service has been done in 2008 by the US–based Software Engineering Institute (SEI), which defined a Web service certification and accreditation process for the US Army CIO/G–6 [207]. This approach was based on a process for certifying web services, which aimed to assure that implemented web services do not expose the Service Oriented Architecture (SOA) infrastructure, in which they are deployed in or interacting with, in order to avoid malicious attacks. Anisetti et al. [15][14] then provide a test-based security certification solution for services and a first approach to its integration within the SOA environment. This approach proposed a solution that defined a hierarchy of security properties to be certified, the classes of tests that could be used to provide the evidence that a set of properties hold for a service, and a matching approach allowing a user to select a service that satisfies their preferences on service certification. Moreover, an important work has been also done regarding the certification for the Quality of Service (QoS) of web services [9]. More approaches, such as [192][194][209], also dealt with defining extended UDDI services, by supporting QoS metadata in the discovery process.

Unlike traditional approaches, the FP7 Project ASSERT4SOA (Advanced Security Service cERTificate for SOA 2010) [23][12] has been focusing on formal and test-based certification of services. It aimed to support new certification scenarios for certifying services and has produced novel techniques, tools, and an architecture for expressing, assessing, and certifying security properties for complex service-oriented applications. These applications are composed of distributed software services that may dynamically be selected, assembled and replaced, and running within complex and continuously evolving software systems. Moreover, it has developed a framework for representing and using machine-readable service security certificates, known as ASSERTS, in service discovery and composition [187][12]

2.4.2.4 MONITORING-BASED CERTIFICATION METHODS AND SECURITY CONTROLS

Several approaches have been developed to support the monitoring of software and cloud services. These approaches support different forms of monitoring, including i) monitoring of individual software services [161][36], ii) service compositions, iii) workflows or orchestrations (e.g., [29][30][161][160][100][214][82][83][113][114][177]), iv) infrastructures for service based systems (e.g., grid and cloud systems [101][179][124][227][106][226][11][2][105][1][26]), v) service level agreements (SLAs) (e.g. [102][160]), or vi) the context of service based systems (e.g., [130][8][35][168][199][200][37]). Most of the cloud monitoring systems focus on performance monitoring without an explicit or adequate focus on security properties (e.g., [105][181][104][101][179][103][84][86][6][85]). There are also, however, systems supporting security monitoring in cloud services (e.g., [218][3][80][79]). Existing approaches can be analysed and classified with respect to a set of key characteristics. These characteristics are related to the language that an approach uses in order to express the monitorable properties, the mode of the monitoring process that it performs (i.e., if it is performed by an external monitor or a monitor embedded in the monitored system), and whether the approach offers diagnosis (i.e., it provide the reasons that have caused the violation) and prediction (i.e., it can detect potential violations of properties or how the value of a monitored property will evolve).

Service monitoring systems have used different types of languages for expressing the properties that they can monitor, including i) formal temporal logic based languages (e.g. [161][160][113][114]), ii) state transition based specifications (e.g., [82][83]), iii) arithmetic specifications (e.g., [36]), iv) assertion languages (e.g., [29][30]), v) modelling languages like UML (e.g., [100][214]), vi) SQL-like languages (e.g., [130]), and vii) scripting languages (e.g. [179]). Moreover, some existing systems perform intrusive monitoring (e.g. [177][130][8][35][168][2][1]). Intrusive monitoring is based on weaving the execution of monitoring activity within the system that is being monitored, such as checking monitoring assertions at specific points within the system's execution code. However, this monitoring systems have three main disadvantages: (a) they introduce a computational overhead into the system that is being monitored, (b) they cannot check all types of properties, such as end-to-end quality of service properties, and (c) they can be disabled by attacks to the very system that they

are supposed to monitor. Other monitoring systems perform non intrusive monitoring, by using an external monitor running in parallel with the system that is being monitored (e.g. [29][30][100][214][82][83][36][113][114][177][226][11][2][105][1]). Non-intrusive monitoring has several advantages over the intrusive monitoring, as it notably reduces the monitoring overhead and increases the resilience to attacks upon the system being monitored. However, these advantages come at a price, as the cost is high for ensuring confidential communication of events to monitors, for the sensitivity of monitoring to attacks over event communication channels, and for delays in violation detection [140][231]. Only a few of the monitoring systems developed for software services offer diagnostic (e.g., [105][232]) and prediction capabilities (e.g. [168][158][157][232]). Monitors are typically integrated with adaptation mechanisms (e.g. [177][11][2][130][8][35][199][200][37]) and in the case of cloud systems, adaptation typically focuses on optimisation of the use of physical and virtual resources (e.g. [103]).

In [80][79], agents positioned at different key points of cloud infrastructure such as VMs of cloud users, VM hosting systems, and data storage components can detect cloud incidents. Examples of systems that focus on performance monitoring include NimBus [181], dynaTrace [105], Ganglia [101], and Nagios [179]. These systems support monitoring of predefined performance metrics (e.g. CPU/memory utilization, pages served per unit time from a server, application calls) for virtual machines and/or cloud applications running in them. From these systems focusing on cloud monitoring, Ganglia [101][167] and Nagios [179] are widely used systems. Ganglia [101] is an open source distributed monitoring system developed to support performance monitoring of clusters and grids. It collects a set of built-in performance metrics (for example CPU, storage and network utilisation metrics) for the different clusters that it monitors and transmits compact cluster states using a multicast-based listen/announce protocol [167]. It also supports user-defined metrics related to resource performance. The use of multicast announcements of the monitoring state of interconnected clusters increases the availability of Ganglia and its resilience to cluster monitor failures but at the same time makes it more vulnerable to attacks as the monitoring state of a cluster can be retrieved from any other cluster in a federation. Nagios [179] is also an open source systems for monitoring IT infrastructure assets including applications, services, operating systems, network protocols, system metrics and infrastructure components using a single environment. Nagios can be configured to perform customised monitoring tasks through shell scripts, executable code in different programming

languages (e.g. C++, Ruby, Python) and APIs. Its scalability and extensibility have made Nagios one of the widely used tools for IT infrastructures monitoring. However neither Ganglia nor Nagios have been developed with a focus on monitoring security properties of clouds. An open source system that has been developed to support the monitoring of security properties and has been applied to software and cloud services is EVEREST [218]. EVEREST adopts a formal language for expressing monitoring properties (based on Event Calculus) that supports the expression of a wide range of properties including basic security properties such as confidentiality, integrity and availability.

A recent development in monitoring software and cloud services is the awareness of the necessity of using monitoring infrastructures that may incorporate multiple and heterogeneous monitors, and which will be capable to configure themselves dynamically without human intervention [95][67][106][26]. This is due to the need of providing uninterrupted monitoring services when the monitoring capabilities, which are available in a service-based system, change due to dynamic changes in the constituent services of such systems [95][67]. The same prerequisite appears also in federated cloud systems [26]. To address these needs the SLA@SOI project has developed a dynamically configurable monitoring infrastructure that can adapt automatically to changes in the monitoring capabilities that are available in service based systems, and that can running on cloud systems and perform dynamic SLA monitorability checks [95][97][103]. The Lattice monitoring system [103] developed as part of the RESERVOIR project also provides support for monitoring dynamically changing federations of cloud systems. However, this approach does not support dynamic SLA monitorability checks, as it focuses only on resource monitoring (CPU, memory and network usage at physical and virtual infrastructure levels) without support for security properties.

Existing approaches in the field of security certification have focused on concrete software components and provide human readable certificates. Thus, they cannot support service-based scenarios that require machine-readable certificates and could support dynamic service selection and composition [72].

Monitoring has also been used at the hypervisor layer to provide incident detection even if the operating system of the guest instance is experiencing critical conditions and monitoring agents

are unable to communicate with monitoring systems. Amazon's CloudWatch [1] is a system of this category. Performance monitoring is also a common function of other cloud hypervisors (e.g., Xen [31], VMWare [221][198]) whilst there are also approaches focusing on monitoring and security of cloud hypervisors (e.g., [198][223][213]). In [198], for example, the workload of virtual machines (VMs) is continually monitored to identify VM security and reliability problems and to allocate resources to the VM accordingly. In [223][222], a hardware-based approach provides isolation of access to shared resources, and in [126] hardware protects VMs memory from unauthorized accesses. The MultiHype architecture [213] allows running multiple hypervisors on a single platform to eliminate security issues.

Dynamic configuration of the monitoring infrastructure aims to provide uninterrupted monitoring services when monitoring capabilities that are available in a service-based system or in a cloud infrastructure change, as a result of dynamic changes in the constituent services of such systems [95][67]. This need appears in federated cloud systems [54]. This type of monitoring infrastructure adapts automatically to changes in the monitoring capabilities that are available in service based systems running on clouds, following dynamic SLA monitorability checks [95][97]. The Lattice monitoring system [103] provides also support for monitoring dynamic changes of cloud federations. Finally, NIST's SCAP specifications [143] and Cloud Security Alliance's Cloud Trust Protocol [56] provide interfaces for extracting monitoring data from cloud systems.

More work has focused on auditing cloud security. The Cloud Controls Matrix (CCM) of the Cloud Security Alliance (CSA) [59], for example, contains a comprehensive set of controls to assess the information security assurance in cloud systems and maps controls to existing frameworks such as PCI DSS [186] and COBIT [62]. CCM is currently being developed through the Open Certification Framework (OCM) [182] into a third party certification program. Moreover, CSA has published the Cloud Audit protocol [58], which provides an automated query interface to cloud services for audit. However, the most relevant certification scheme based on continuous monitoring is the CSA STAR Level 3 [60], which is still under development.

2.5 SUMMARY OF CERTIFICATION SCHEMES FOR CLOUD SERVICES

Table 6 below summarises the different certification schemes for cloud services regarding the way evidence is being collected for certifying cloud service providers, the Life Cycle Model defined as well as the security properties that can be certified by each scheme.

Table 6 - Cloud Certification based on Evidence Collection

Summary of Certification Approaches in Cloud Services				
Approach	**ToC**	**Evidence Type**	**Life Cycle Model**	**Security Property Categories**
ComCert [64]	Cloud Services	(Automated) Inspection	Explicit	- Application & Interface Security (Auditability), - Interoperability & Portability, - Legal & Standards Compliance
CIF Code of Practice [55]	Cloud Services	Questionnaire	Explicit	- Data Security & Information Lifecycle Management, - Business Continuity Management & Operational Resilience
COBIT [62]	Cloud Services	Inspection	Explicit	- Governance and Risk Management, - Identity & Access Management
FedRAMP [92]	Cloud Services	Inspection	Explicit	- Identity & Access Management, - Business Continuity Management & Operational Resilience, - Legal & Standards Compliance
McAfee Cloud Security Platform [169]	Cloud Services	Inspection / Monitoring	Explicit	-Application & Interface Security, - Identity & Access Management, - Legal & Standards Compliance, - Data Security & Information Lifecycle Management, - Data Centre Security
TRUSTed Cloud Certification Platform	Cloud Services	Inspection	Explicit	-Application & Interface Security, - Encryption & Key Management, - Legal & Standards Compliance

CSA STAR - Level 1 [60]	Cloud Services	Questionnaire	Explicit	-Application & Interface Security, - Identity & Access Management, -Legal & Standards Compliance, - Data Security & Information Lifecycle Management, -Business Continuity Management & Operational Resilience
CSA STAR - Level 2 [60]	Cloud Services	Inspection / Questionnaire	Explicit	
CSA STAR - Level 3 [60]	Cloud Services	Monitoring	Undefined	

2.6 SUMMARY

In this chapter the related work done in the area of certification has been presented. There is a substantial work done for certifying services and different controls have been deployed to collect evidence to support the transparency of the process. However, according to the literature review, current certification models for cloud services are not yet fully deployed to handle the effects that different types of cloud mechanisms used can have on security properties. Moreover, there are not any clear objective security assessment schemes to express: i) factors that should be taken into account for assessing cloud security properties, ii) specific criteria that can be used to assess these factors, iii) the evidential bases for the assessment of these criteria, and iv) the combination of assessments for individual criteria into an overall scheme.

Furthermore, there are not yet technical solutions to certify the security and trustworthiness of cloud-based services and applications, in order to guarantee an appropriate level of assurance, integrated with existing cloud infrastructure. Finally, the existing certification schemes are not able to automatically identify appropriate evidence acquisition plans and security assessment models.

With regards to the monitoring based cloud certification, which focuses on monitoring the resources (e.g., CPU, memory and network usage) for resource optimisation and elasticity in security compliance, existing cloud monitoring solutions do not support at all or provided limited support for this type of assessment in cloud systems.

Thus, based on these limitations of the existing work in the field, this thesis defines a certification model, in order to enable an objective assessment of security properties using monitoring solutions for gathering evidence, regarding different factors of cloud security. It will also provide a single monitoring infrastructure that will be able to monitor security properties and resources, tailored to the need of users, in order to provide adequate monitoring evidence for the purposes of the certification process. Finally, it will define a framework that can support an automated certification process based on continuous monitoring, for certifying cloud-based services and applications.

Chapter Three

BACKGROUND TECHNIQUES

3.1 OVERVIEW

This chapter provides information about the two frameworks that have been used to support the development and validation of the monitoring based certification framework introduced in this thesis, namely the runtime-monitoring framework EVEREST and the probabilistic model-checking framework Prism [191]. More specifically, our approach will use an event-monitoring framework, called Everest (EVEnt RESoning Toolkit [218]), which is runtime-monitoring framework that is based on first order temporal logic language of Event Calculus [210].

Section 3.2 provides a short overview on Event Calculus language and Section 3.3 provides an extended discussion of the monitoring framework highlighting on the formal specifications it uses. Finally, Section 3.4 covers the principles of the Prism model checker that underpins our verification of the certification process.

3.2 EVENT CALCULUS

3.2.1 OVERVIEW

Event calculus (EC) is first order temporal logic language for representing events and their effects and has been used to represent and reason about the behaviour of dynamic systems [210]. EC is based on first-order predicate calculus [210].

In the following, we give a brief overview of EC, as it provides the logic foundation of the reasoning processes that underpin EVEREST, i.e., the runtime monitoring system that has been

used to realise the certification framework that is introduced by this thesis [218][216]. In particular, EC provides the semantic foundation of *EC-Assertion*, i.e., the monitoring language of EVEREST and the language that we have defined in our framework in order to specify security assertions as part of monitoring based certification models. The monitor that we are using in our framework, the EVEREST monitor, uses the language that specifies a monitorable property in terms of events and fluents.

3.2.2 SPECIFICATION OF EC FORMULAS

EC formulas are first order temporal logic formulas expressed in terms of events, fluents and time points. A fluent is anything whose value is subject to change over time [210]. For instance, fluents can be conditions regarding the state of a system at a particular instance of time. Fluents are initiated and terminated by events. An event in EC is something that happens at a specific instance of time, such an invocation of an operation. It has instantaneous duration and may change the state of a system.

EC allows the specification of this type of system events, as well as the time when they occur. It also allows the specification of initialisations and modifications of system that these events might cause at specific times. The basic predicates that the EC language uses are presented in the Table 7 and explained below.

To represent the occurrence of events, EC uses the predicate *Happens(e,t)*. This predicate signifies that an event *e* occurs at time *t*. To initiate a fluent is, the EC uses the predicate *Initiates (e,f,t)*, which means that a fluent *f* starts to hold after the occurrence of the event *e* at time *t*. The termination of a fluent is indicated by predicate *Terminates(e,f,t)*, stating that a fluent *f* ceases to hold after the occurrence of the event *e* that occurs at time *t*. An EC formula may also use the predicates *Initially(f)* to state that a fluent *f* holds at the start of the operation of a system and the predicate *HoldsAt(f,t)* that states that a fluent *f* holds at a specific time *t*. All predicates are summarised in Table 7.

Table 7 - EC Predicates

Happens(e,t)	An event *e* occurs at time point *t*
Initiates (e,f,t)	A fluent *f* holds after an event *e* has occurred at time point *t*
Terminates(e,f,t)	A fluent *f* cease to exist after an event *e* has occurred at time point t
Initially(f)	A fluent *f* holds from time *0*
HoldsAt(f,t)	A fluent *f* holds at time point *t*

Event calculus defines a set of axioms that can be used in order to determine when a fluent holds based on its initiation and termination by the occurrence of different events. These axioms are listed in Table 8 below.

Table 8 – Axioms of Event Calculus

EC.1: ∃ e,t,
Happens(e,t,R(t1,t2))
∧ Terminates(e,f,t)
⇒ Clipped(t1,f,t2)

EC.2: ∃ e,t
Happens(e,t,R(t1,t2))
∧ Initiates(e,f,t)
⇒ Declipped(t1,f,t2)

EC.3: Initially(f)
∧ ¬Clipped(0,f,t)
⇒ HoldsAt(f,t)

EC.4: ∃ e,t1
Happens(e,t1,R(t1,t2))
∧ Initiates(e,f,t1)
∧ ¬Clipped(t1,f,t2)
⇒ HoldsAt(f,t2)

EC.5: ∃ e,t1
Happens(e,t1,R(t1,t2))
∧Terminates(e,f,t1)
∧ ¬Declipped(t1,f,t2)
⇒ ¬HoldsAt(f,t2)

EC.6: HoldsAt(f, t1)
∧ t1 < t2
∧ ¬Clipped(t1,f,t2)
⇒ HoldsAt(f,t2)

EC.7: ¬HoldsAt(f, t1)
∧ (t1 < t2)
∧ ¬Declipped(t1,f,t2)
⇒ ¬HoldsAt(f,t2)

The first axiom EC.1 states that a fluent *f* is clipped, meaning that it ceases to hold, at some time point within the time range between *t1* and *t2*, if an event *e* occurs at some time point *t* within this range, which will cause the termination of the fluent *f*.

The axiom EC.2 states that a fluent *f* is declipped, which means that it comes into existence, at some time point *t* within the time range between *t1* to *t2*, if event *e* occurs at that time point *t*, within the range *t1* and *t2*, which initialised the fluent *f* at time point *t*.

The third axiom EC.3 states that a fluent *f* holds at the time point *t*, if it was already held at time *0* and has not been terminated by any event between the time range between *0* and *t*.

The fourth axiom EC.4 states that a fluent *f* holds at the time point *t2*, if an event *e* has occurred at some time point *t1* and before the time point *t2,* which had initiated the fluent *f* at that time point *t1*, and *f* has not been clipped between this time range between *t1* and *t2*.

The fifth axiom EC.5 states a fluent *f* does not hold at the time point *t2*, if there was an event *e* that occurred at some time point *t1* before *t2*, which terminated the fluent *f* and this fluent has not been declipped at any time point between the range *t1* and *t2*.

The sixth axiom EC.6 states that a fluent *f* holds at a time point *t2*, if it was held at time point *t1*, where time point *t1* is before the time point *t2*, and if the fluent *f* has not been clipped between the time range between *t1* and *t2*.

The seventh axiom EC.7 states that a fluent *f* does not hold at a time point *t2*, if it was not held at some time point *t1* before *t2* and if *f* has not been declipped since then.

3.3 EVEREST

EVEREST (Event Reasoning Toolkit) is a run-time monitoring framework, which aims to track the runtime behaviour of a system and to identify whether it satisfies certain properties required of it. This activity takes the specification of the properties required of a software system as input and checks whether the traces of events, which are produced by the system at runtime, are consistent with the properties [218].

There are many reasons why we have selected EVEREST as the underlining system of our monitoring system. The main reason was because it uses the EC-Assertion language, which is a powerful language in terms of expressiveness of temporal first order language, and it can be used to specify a wide range of system properties. Moreover, EC-Assertion provides an explicit language for expressing time conditions in terms of linear formula constraints. Another reasons was the ability of EVEREST to express security and dependability properties and its ability to support the monitoring of events that may have been produced by distributed systems using different clocks, as discussed in [140]. More work on supporting the checks of monitorability and dynamic set up of EVEREST monitoring configurations was presented in [67][95], leading us to have more reasons to use EVEREST for supporting our approach.

EVEREST is an open source system that has the architecture is shown in Figure 3. As explained in [218], EVEREST is exposed as a service and is used to offer interfaces in order i) to submit monitoring rules to it for checking, ii) to forward runtime events from the applications that are being monitored, and iii) to obtain the monitoring results from it. The main three components of EVEREST are: i) a monitor manager, ii) a monitor, and iii) an event collector.

The *monitor manager* is responsible for initiating, coordinating and reporting the results of the monitoring process. In order to do so, it first receives the monitoring rules through the *CheckRules* API and then it provides an *IMonitor* API for obtaining the monitoring results. The *event collector* is the component that is responsible for receiving events from the service in order to then pass them to the monitor manager component. Afterwards, the monitor manager component forwards these events to the Native Type Generator (NTG) sub-component of the monitor, which is responsible to translate these events from the XML format to internal Java objects. After receiving the events from the monitor manager, the monitor checks if they satisfy or

violate any of the given monitoring rules and it returns the monitoring results to the monitor manager.

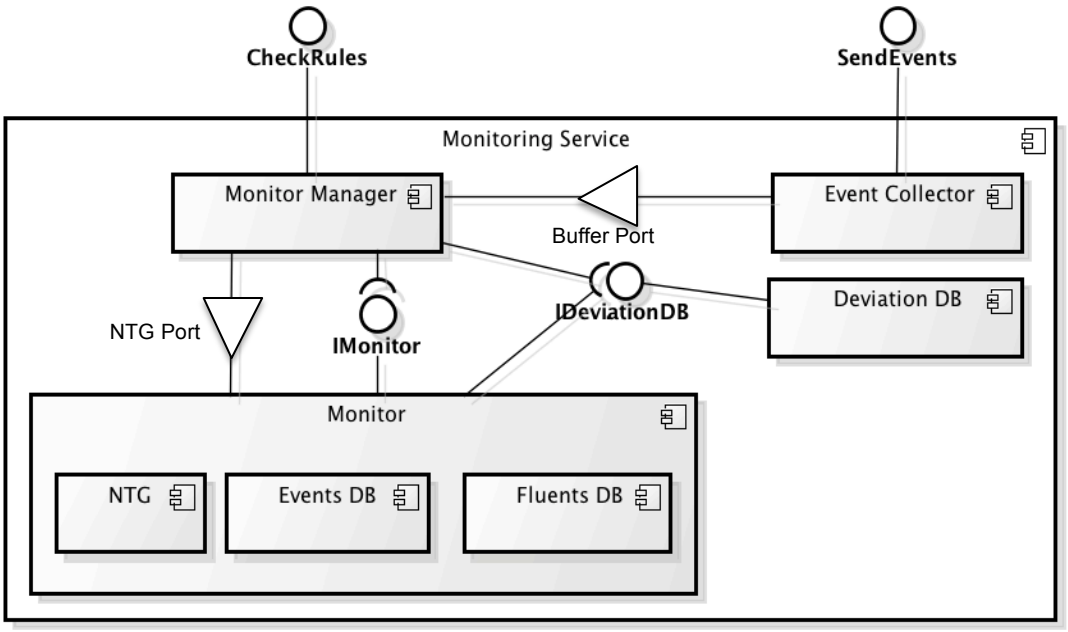


Figure 3 – EVEREST Architecture (Source [218])

EVEREST is a general-purpose engine for monitoring behavioural and quality properties of distributed systems based on events captured from them during the operation of these systems at runtime. The properties that can be monitored by EVEREST are expressed in an Event Calculus [210] based language called EC-Assertion, which is a language based on the EC language that was explained in the previous section.

EVEREST uses two different types of formulas for the monitoring process, namely monitoring rules and assumptions. The formulas for the monitoring rules express the properties that need to be checked at runtime and have the form of "body ⇒ head". The meaning of a monitoring rule is that if its body evaluates to "True", its head must also evaluate to "True". EC-Assertion also uses monitoring assumptions, which have the same form as rules but their meaning is that when their body evaluates to "True", their head can be deduced. Thus the formulas for the assumptions are used in order to derive information about the state of the system that is being monitored based on

observations of its behaviour and the state of the monitoring process itself. Furthermore, as we will explain later, assumptions could also be used for the identification of possible anomalous behaviour or conflicts of the monitoring rules, to detect and prevent possible attacks.

An example of the basic predicates used by EC-Assertion for expressing events and fluents for the availability security property, and their significance are presented in Table 9, which were also presented and explained in [145]. Service availability is defined as the ratio of the period during which a service is unavailable over the total period of monitoring a service. A period of unavailability is defined as the period between a time point when a call to a service operation is not served and the next time point in the future at which a call to an operation of the same service is served. The parametric templates are defined in terms of *request* and *response* of events in EC-Assertion, and there are three fluents specified for the computation of the availability.

Table 9 - Monitoring template for Availability

***⟨Availability⟩*tdef==**

A0.Availability.<CaseId>: Initially(LastServiceMonitoringPeriod(<_SrvId>, systemTime())

A1.Availability.<CaseId>:
Initially(UnavailablePeriods(<_SrvId> ,_PN, _P[]))

A2.Availability.<CaseId>:
Happens(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧
¬**Happens**(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧
¬∃ _PN, _ST,: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧
∃ _PN, _P[]: **HoldsAt**(UnavailablePeriods(<_SrvId>, _PN, _P[]), t1)) ⇒
Initiates(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), Unavailable(_PN+1, <_SrvId>, t1), t1) ∧
Terminates(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, _P[]), t1) ∧
Initiates(e(_id1,_Snd,<_SrvId>,Call(_O),<_SrvId>),UnavailablePeriods(<_SrvId>,_PN+1,_P[]), t1)

A3.Availability.<CaseId>:
Happens(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧

Happens(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧

∃ _PNum, _ST: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1) ⇒

Terminates(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), Unavailable(_PN, <_SrvId>, _ST), t1+1)

A4.Availability.<CaseId>:

Happens(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧

Happens(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧

∃ _PN, _ST,: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧

∃ _PN, _P[]: **HoldsAt**(UnavailablePeriods(<_SrvId>, _PN, _P[]), t2)) ⇒

Terminates(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, _P[]), t2) ∧

Initiates(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, append(_P[], t1 – ST)), t2)

R.Availability.<CaseId>:

Happens(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧

Happens(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧

∃ _PN, _ST, _P []: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧

HoldsAt(UnavailablePeriods(<_SrvId>, _PN, _P[]), t2) ∧

HoldsAt(LastServiceMonitoringPeriod(<_SrvId>, _lmsTime), t2)) ⇒

sum(_P[]) / (t2 – _lmsTime) > K

These fluents are:

1. ***Unavailable(_PN, <_SrvId>, _ST)***

This fluent is used to keep track of the unavailability of a service and has three parameters:

- The first parameter (*_PN*) counts the number of unavailable periods for a monitored service,
- The second parameter (<*_SrvId*>) records the unique ID of the monitored service, and
- The third parameter (*_ST*) records the time point when the service becomes unavailable.

2. UnavailablePeriods(<_SrvId> ,_PN, _P[]))

This fluent is used to keep track of unavailable periods of a service and has three parameters:

- The first parameter (i.e. <_SrvId>) records the unique ID of the monitored service,
- The second parameter (i.e. _PN) records the count of unavailable periods of the monitored service and
- The third parameter (i.e. _P[]) records duration of each unavailable period of the monitored service.

3. LastMonitoringPeriod(<_SrvId>, _lmsTime), t2)

The fluent is defined to record the starting time point of the monitoring session and has the following two parameters:

- The first parameter (i.e. <_SrvId>) records the unique ID of the monitored service, and
- The second parameter (i.e. systemTime()) signifies a standard system call that is executed by the monitor in order to obtain the current time of the system where the monitoring service is running.

The assumption *A0* and *A1* initiate the *LastMonitoringPeriod* and *UnavailablePeriods* fluents for first time. The assumption *A2* starts a new period of unavailability when a non-served event occurs and increases the number of the unavailability periods. The assumption *A3* terminates current period of unavailability for a service, when a served event occurs. The assumption *A4* records the length of a terminated period of unavailability. Finally, the rule *R* checks if the availability of a service is greater than K.

In the availability template form, the <*_SrvId*> is the unique identifier of the service that the availability is checked for, and the <*CaseId*> that the template is selected for and refers to, is the unique id of the certificate, which was assigned to EVEREST. It should be noted that EVEREST would receive primitive call and response events from the service, thus a call to the monitored service occurred at *t* is considered to be served if a corresponding response occurs within a predefined time range between *t* and *t+d*. The value of d is denoted as <*D*> in the templates. During the translation process concrete values of <*_SrvId*>, <*CaseId*> and <*D*> are chosen according to a predefined set of criteria.

3.4 PRISM

PRISM [146][191] is a probabilistic model checker, i.e., a tool for formal modelling and static analysis of systems that have random or probabilistic behaviour. It has been used to analyse systems from many different application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others. The reason we decided to use the Prism model checker was mainly because of its ability to process simulations without probabilities combined with the ability to also model probabilistic system behaviour. Furthermore, it has an advanced tool support and an extensive documentation of its use and its theoretical foundations, which facilitated the set up and the execution of the model checking evaluation of this thesis.

PRISM can build and analyse several types of probabilistic models, such as:

- ***Discrete-Time Markov Chains (DTMCs)***

DTMS are state-transition systems augmented with probabilities. In these systems a discrete set of *states* should be defined to represent all possible configurations of the system being modelled [147]. Moreover, the transitions between states should also be defined, which occur in discrete time-steps. Every state should have at least one outgoing transition or a self-loop to represent a final or terminal state, in order to avoid deadlock states. Finally, the probabilities are also defined, which define the probability of choosing a specific transition between states. This probability is given by discrete probability distributions.

The DTMC is a tuple of the form (S, S_{INIT}, P, L), where S is a finite set of states, $S_{INIT} \in S$ is the initial state, P: $S \times S \rightarrow [0,1]$ is the transition probability matrix where $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$, and L: $S \rightarrow 2^{AP}$ is a function to label states with atomic propositions [147].

A simple example of this type of probabilistic model is shown in the following figure.

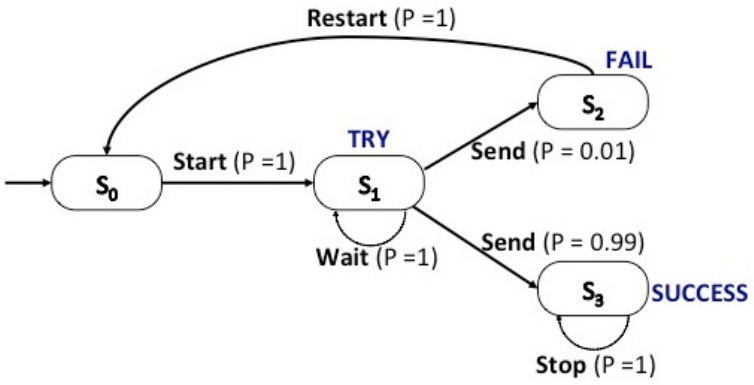


Figure 4 – DTMC simple example

As shown in the example, the initial state is S_0. After the first transition called "*Start*", which will always take place because it has probability of occurrence 1, from state S_0 to S_1, the process will try to send a message. From S_1 there is a probability of 0.01 to wait in case the channel for sending a message is not ready, and thus the process will wait a step (self-loop at S_1), there is a probability of 0.98 to send a message successfully (move to S_3) and stop and a probability of 0.01 to fail sending the message (move to S_2) and restart (move back to S_1).

Therefore, according to the DTMC tuple:

D = (S, S_{INIT}, P, L)
S = {s0, s1, s2, s3}
S_{INIT} = s0

AP = {try, fail, success}
L(s0)= 0,
L(s1)={try},
L(s2)={fail},
L(s3)={success}

- ***Markov Decision Processes (MDPs)***

MDPs extend the DTMC by combining discrete probabilities and non-determinism for accurate modelling of concurrency [147]. Non-determinism enables the modelling of asynchronous parallel composition of probabilistic systems, and permits the specification of certain aspects of a system.

The MDP is a tuple of the (S, s_{INIT}, Steps, L) where *S* is the set of states, s_{INIT} is the initial state, *L* is the labelling function, and Steps: $S \rightarrow 2^{Act \times Dist(S)}$ is the transition probability function where *Act* is a set of actions and *Dist(S)* is the set of discrete probability distributions over the set of states *S*.

- ***Continuous-Time Markov Chains (CTMCs)***

CTMC specify the rates $\rho(s, s_0)$ for making a transition from states to s_0, with the interpretation that the probability of moving from s to s_0 within t time units is $1 - e^{-\rho(s, s_0)*t}$. It has discrete state space, continuous time, and exponentially distributed delays. It is usually used to modelling component lifetimes, inter-arrival times, or biochemical reaction rates.

- ***Probabilistic Automata (PAs)***

PAs probabilistic extension of timed automata and consists of discrete states, real-time clocks, discrete probability distributions, and non-determinism.

Models are described using the PRISM language, which is a simple, state-based language. PRISM provides support for automated analysis of a wide range of quantitative properties of these models. The property specification language incorporates the temporal logics PCTL (Probabilistic Computation Tree Logic), CSL, LTL and PCTL*, as well as extensions for quantitative specifications and costs/rewards [191].

PRISM incorporates state-of-the art symbolic data structures and algorithms, based on BDDs (Binary Decision Diagrams) and MTBDDs (Multi-Terminal Binary Decision Diagrams) [148][185]. It also includes a discrete-event simulation engine, providing support for approximate/statistical model checking, and implementations of various different analysis techniques, such as quantitative abstraction refinement and symmetry reduction [146].

Chapter Four

MONITORING BASED CERTIFICATION PROCESS AND MODEL

4.1 OVERVIEW

The aim of this chapter is to provide the reader with a description of the certification process based on continuous monitoring, and the certification model that needs to be submitted at the proposed framework. It presents and explains a high level overview of the certification process based on continuous monitoring and gives definitions to some concepts used. Moreover, it provides the specification of the proposed Certification Model, which is expressed in the XML language, thus the XML schema used to describe the CM is presented, as well as all its elements and sub-elements that need to be defined.

4.2 MONITORING BASED CERTIFICATION PROCESS

The advantages of dynamic software service provision on cloud systems needs to ensure that the software systems and infrastructures that underpin these services have the appropriate assurance levels for the intended purpose. A key prerequisite for this assurance is to ensure that these services and the underlying systems continuously satisfy a set of security properties that are necessary for this assurance.

A monitoring based certification process is defined as a process where the assessment of a security property regarding a specific cloud service is based on operational evidence from the provision of that service gathered through continuous monitoring. The output of this process is a type of certificates, called *monitoring-based certificate*. Monitoring-based certificates are digital

certificates that contain conditions that refer to the actual state of the cloud service at the moment of the use of the certificate, as opposed to current certifications that can only refer to conditions verified at the moment of their production. This type of certificates is significant for dynamic environments like Cloud Computing, which allows a high degree of continuous assurance for the cloud services and applications, at run time.

An important step towards generating the monitoring based certificates is to specify a certification model (CM) defining the process of generating and managing digital security certificates in manner that will enable the automatic execution of this process. As stated in Chapter 2, a Certification Model (CM) is used to define the process of certifying cloud services, i.e., the process of assessing continuously whether cloud services satisfy required security properties, producing certificates asserting that they do so, and managing these certificates.

In line with classic approaches to software certification as described in Section 2.4.1.1, the production of certification models is the responsibility of certification authorities. Such authorities can use a certification model to specify processes that reflect their specific technical approach to certification, business processes (e.g., in as far as the management of certificates is concerned) and/or industry or other regulatory standards about security properties and their assessment.

To fulfil the above purpose a CM should include:

- The security property that needs to be certified, such as *AIS:non-repudiation:non-repudiation-of-origin*, which consists of:
    a. the security property category that belongs to, as explained in Section 2.3.1. For example *Non-Repudiation* belongs to the *Application & Interface Security property (AIS)* category; and
    b. the assertion, that defines the formal operational definition of the property that will be used for monitoring, in this example *non-repudiation:non-repudiation-of-origin*
- the ToC that this property applies to, which is the actual service that needs to be certified,

- an assessment scheme that determines the evidence that should be taken into account for assessing a property,
- how frequently this evidence should be checked,
- when the accumulated evidence will be sufficient for issuing the certificate, and
- the certificate life cycle, i.e., the states which a digital certificate may be at (e.g., activated, issued, revoked) and how it may transform itself between them, and
- any additional validity checks that will need to be satisfied at different states of the certification process (e.g., checks that the software components used to collect and process the evidence acquired for the cloud services have maintained their integrity during the process, i.e., they have not been tampered with).

Moreover, when defining the security property to be certified, its assertion should also be defined that states the set of rules concerning the evidence collection for a given security property. Consequently, we can state that the input for the certification process is: a) the definition of the Security Property and the specification of its measurements, and b) the Assertion, which contains attributes to support the evidence collection. However, it needs to be mentioned that the precise semantics of the Security Property in a CM are determined by the monitoring rules included.

The assessment scheme is another element of the CM that defines general conditions regarding the evidence that must be collected, in order to be able to issue and maintain a certificate, according to the particular certification model that a certifier has defined. These conditions define the expiry date of the certificate, the frequency of the evidence collection and conditions regarding possible anomalous behaviour or conflicts.

Finally, the certificate life cycle should also be defined in the CM. A Certificate Life Cycle is a state chart diagram that defines all possible states a certificate can take during its life. It also defines the transition between different states that refer to specified conditions of the Assessment Scheme in the CM. The certificate life cycle will be used as a guideline for the certificate models, since it will provide basic conceptual entities like revocation and validation. More specifically, a

certificate may have different states in its life cycle (e.g. Issued, Revoked, Renewed etc.) and different transitions that lead to these states. Each transition refers to a specified condition that should be met, which will lead to specific actions concerning a certificate and its behaviour.

In the certification model a certification authority (CA) will be able define its own Certificate life cycle model, with different number of life cycle states and transitions. For example, when the evidence gathered for a certificate type is sufficient for verifying the security property related to it (as determined by the certification model), a certificate that is an instance of this type of CM could be issued. However, even after they are issued, certificates can be updated subject to changes in the operational conditions of the cloud service that they are associated with. The possible updates and any other key changes in the life cycle of monitoring based certificates are described in a certificates' life cycle model element.

Since the security and trustworthiness of cloud services requires continuous and transparent assessment, our approach proposes a monitoring based certification process for cloud services. This process, in an abstract level description, involves the following main steps:

- The submission of a CM from a CA,
- The translation of the Security Property assertion from the XML language used in the CM into EC-Assertion language of the monitor,
- The extraction and capturing of the evidence about the operation/provision of services from the cloud infrastructure,
- The continuous communication of this evidence to a monitor that checks the security property assertions, as specified in the CM,
- The collection of the monitoring results of the security property for the generation of monitoring based certificates, when the evidence is positive and sufficient,
- The update of certificates status, based on the evidence and the conditions defined in the CM, and

- The provision of all evidence from the process for auditing purposes, to support the transparency of the process.

According to the above steps, the certification process starts when a CA requires certifying a service, by submitting a complete certification model, where all relevant information concerning the process is being defined. Subsequently, the framework after receiving the CM, it translates the security property assertions to EC-Assertion formulas to pass them to the EVERST monitoring tool, to start the monitoring process of the TOC. When the process initiates, the monitor starts receiving evidence from the TOC, at runtime, and checks them according to the defined assertion rules and sends the monitoring result to the framework. The framework will then check if the conditions defined in the CM are satisfied based on the monitoring results that it received from the monitor, and according to the life-cycle model will update the status of the certificate. For example, if enough evidence is received and they satisfy the security property rule, then a certificate is issued. Moreover, all evidence collected by either the monitor or the TOC are stored in the framework, in case an auditor requires checking them.

4.3 CERTIFICATION MODEL FOR MONITORING BASED CERTIFICATES

In monitoring based certificates, the evidence required for assessing and verifying security properties is acquired through continuous monitoring of the cloud services' operations. Hence, the evidential for such certificates can cover contextual conditions that might not be possible to predict, test or simulate through other forms of assessment, such as testing or static analysis, that take place before the deployment of a cloud service.

As with all types of certificates, the process of generating and managing monitoring based certificates is driven by certification models. The purpose of such models is to define the security property that needs to be certified, the types and extent of evidence that should be acquired in order to be able to certify the property, the life cycle of certificates of the given type, and the agents which will have the responsibility to carry out different parts of the process.

To enable the definition of certification models for monitoring based certificates, we have developed an XML schema for specifying all the required elements for generating this type of certificates. More specifically, the schema provides means for specifying the security property to be certified, an assertion providing a formal definition of this property, the certification authority who will sign of the certificate, and the conditions that need to be followed in order to generate or update the status of the certificates.

4.3.1 CERTIFICATION MODEL XML SCHEMA DESCRIPTION

The top layer of the proposed certification model schema for specifying Monitoring based certification models is shown in Figure 5:

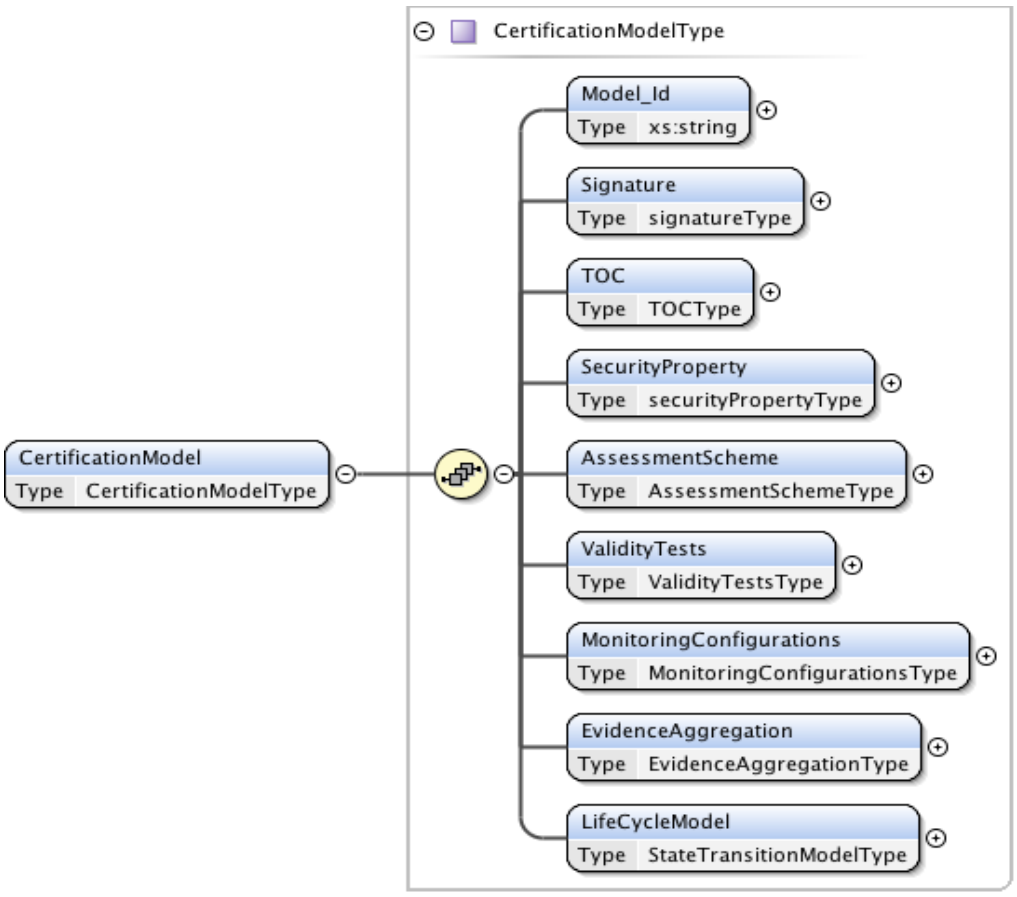


Figure 5 – Certification Model Schema Elements

As shown in the figure, above, a CM consists of the following elements:

1. *Model_Id*
2. *Signature*
3. *ToC*
4. *Security Property*
5. *Assessment Scheme*
6. *Validity Tests*

7. *Monitoring Configurations*
8. *Evidence Aggregation*
9. *Life Cycle Model*

Below, each element of the model is explained in details, as well as their meaning and purpose is defined.

4.3.1.1 MODEL_ID ELEMENT

The first element of the certification model is the *Model_Id*, which represents the unique identifier of the certification model instance. *Model_Id* is an element of type integer. An example of model id is shown below:

```
<Model_Id>1001</Model_Id>
```

It should be noted that the identifier of the certification model is different from the identifier of an instance of the model that is used when the model is applied, in order to certify a given property of a particular ToC.

4.3.1.2 SIGNATURE ELEMENT

Signature is the element in the certification model that represents the signature of the certification authority that has defined it. This is not a binary signature as current digital certificates have, but is an element to identify the certifier that is responsible for the validity of the certificate, which they sign. As shown in the figure below, *Signature* is an element of type *signatureType* and has two sub-elements, which are:

- the *Name* element, which defines the name of the certifier, and
- the *Role* element, which defines the role of the certifier. Both sub-elements are of a type *string*.

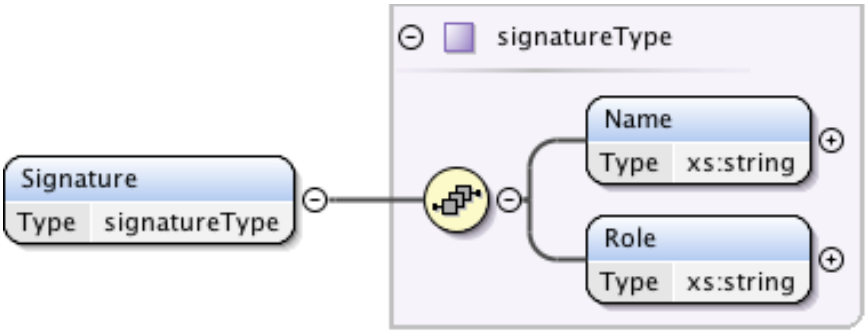


Figure 6 – Signature Element

An example of the *Signature* element is provided below.

```
<Signature>
   <Name>City</name>
   <Role>CA</Role>
</Signature>
```

4.3.1.3 TARGETOFCERTIFICATION (TOC) ELEMENT

The *TargetOfCertification* (ToC) element describes the cloud service that needs to be certified by the particular instance of the certification model. ToC is an element of type *TargetOfCertificationType*.

As shown in the Figure 7 below, the specification of a ToC includes:

- An attribute, called "*id*", which represents the unique identifier of the ToC. This attribute is mandatory, and

- A sequence of *providesInterface* and *requiresInterface* elements. These interface elements specify sets of operations whose execution and/or results will need to be monitored during the certification process.
    - The *providesInterface* elements specify the interfaces that the ToC offers itself.
    - The *requiresInterface* elements specify interfaces that the ToC expects an external entity to have.

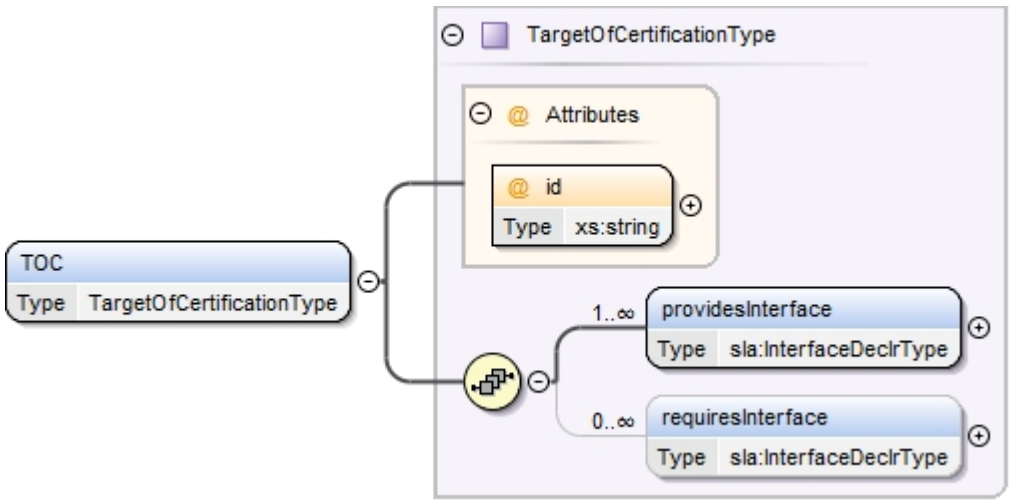


Figure 7 – TargetOfCertification Type

The XML schema for specifying *TargetOfCertification* elements is given in below:

```
<xs:complexType name="TargetOfCertificationType">
  <xsd:sequence>
    <xsd:element minOccurs="1" maxOccurs="unbounded" name="providesInterface"
                 type="sla:InterfaceDeclrType"/>
    <xsd:element minOccurs="0" maxOccurs="unbounded" name="requiresInterface"
                 type="sla:InterfaceDeclrType"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
```

The two sub-elements of a ToC that specify the interfaces that should be used and are both of the type *InterfaceDeclrType*. These are:

- The *providesInterface* sub-element

  As shown in Figure 8, this sub-element includes a sequence of texts and properties that define the interface that a ToC itself realises. The *providesInterface* defines what operations will be invoked.

- The *requiresInterface* element

  As shown in Figure 9, this sub-element includes a sequence of i) *ID*, ii) provider references (*ProviderRef*), iii) zero or more *Endpoints* and iv) *Interfaces*. This element defines the interfaces that the ToC requires from external entities, in order to be able to realise the functionality that will be monitored during the certification process. Thus, for these interfaces it is important to specify the endpoint where the relevant operations can be invoked.

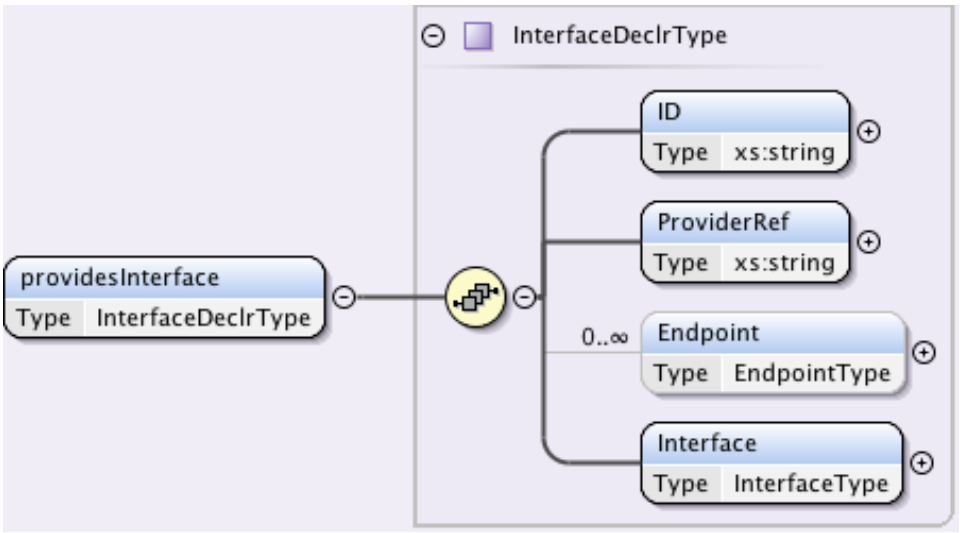


Figure 8 – ProvidesInterface Type

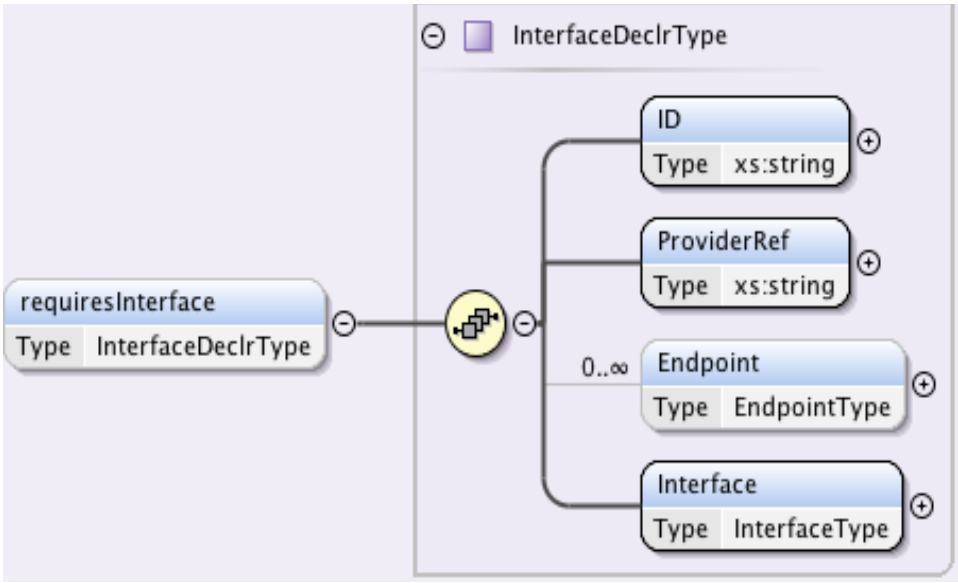


Figure 9 – RequiresInterface Type

4.3.1.4 SECURITYPROPERTY ELEMENT

SecurityProperty is the element in the schema that defines the security property that is to be certified by the specific instance of the certification model. *SecurityProperty* is an element of type *SecurityPropertyType*. As shown in Figure 10, this type has:

- Four attributes, called "Security*PropertyId*", "*SecurityPropertyDefinition*", "*Vocabulary*" and "*ShortName*", which are used to define the property to be certified, such as availability, integrity, confidentiality, etc.,
- A sub-element called *sProperty* of a type *propertyType*, which is used to define more concrete the property to be certified, and
- A sub-element, called *Assertion*, which is used to provide the definition of the security property to be certified. *Assertion* is an element of complex type *AssertionType*. To define this type we have created a new XML based language to be able to express the specification of security properties (e.g., the introduction of multi-valued variables, event timestamps, forced executions of actions)

The specification of the *SecurityProperty* element in XML schema is listed below.

```
<xs:complexType name="securityPropertyType">
  <xs:sequence maxOccurs="1" minOccurs="1">
    <xs:element name="sProperty" type="propertyType"/>
    <xs:sequence minOccurs="0">
      <xs:element name="Assertion" type="AssertionType"/>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="SecurityPropertyId" type="xs:string" use="required"/>
  <xs:attribute name="SecurityPropertyDefinition" type="xs:string"
                use="required"/>
  <xs:attribute name="Vocabulary" type="xs:string"/>
  <xs:attribute name="ShortName" type="xs:string"/>
</xs:complexType>
```

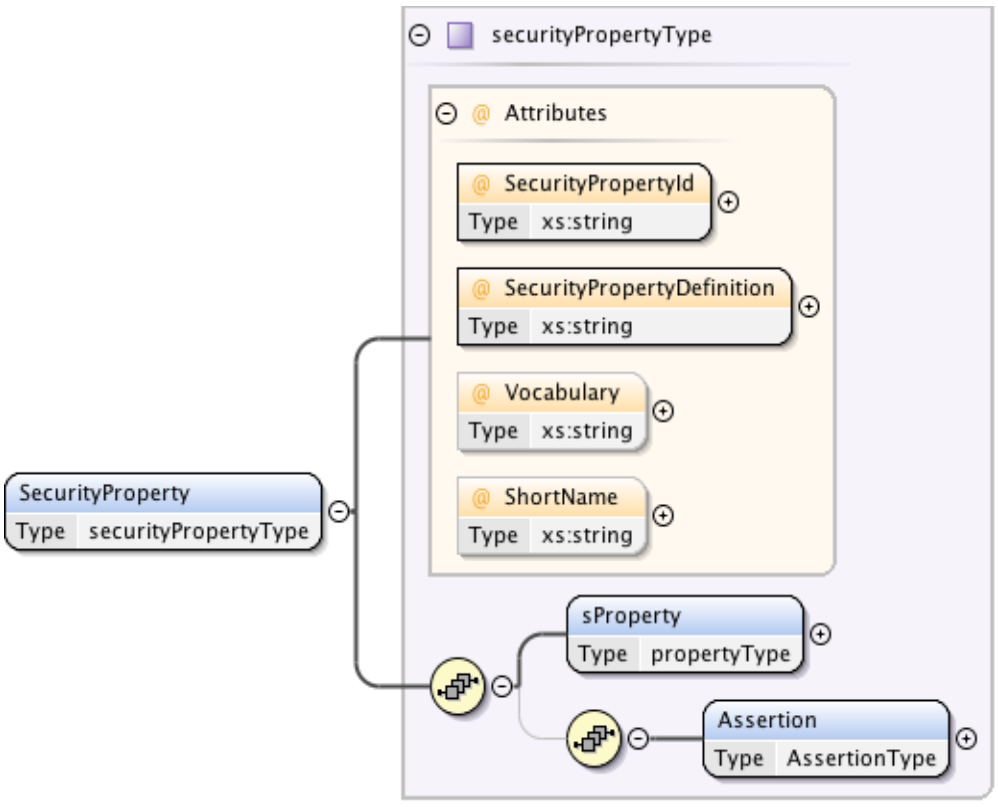


Figure 10 – SecurityProperty Element

Within a monitoring based certification model, the security property to be certified for TOC is specified by one or more monitoring rule and zero or more assumptions:

Security-property:= MonitoringRule* [“,” MonitoringAssumption]*

In a security property specification, monitoring rules are assertions expressing conditions that must be satisfied during the monitoring of TOC, whilst monitoring assumptions are assertions, which are used to record and update state variables indicating the state of TOC during monitoring. Both monitoring rules and assumptions are expressed as assertions in EC-Assertion+. EC-Assertion+ is an extension of EC-Assertion, i.e., the language for expressing monitoring conditions in the EVEREST monitoring system [218], which is part of the CUMULUS framework. EC-Assertion+ is based on Event Calculus [210]. Within it, assertions are formulas of the form:

Assertion ::= [precondition]* “⇒” postcondition

The (optional) precondition element in an assertion determines the conditions under which the assertion should be checked. The meaning of the postcondition element depends on whether the assertion is a monitoring rule or an assumption. In assertions expressing monitoring rules, postcondition determines the conditions that are guaranteed to hold (i.e., should be true if the preconditions are true). In assertions expressing monitoring assumptions, postcondition determines the states of the system that can be inferred to be true if the preconditions are true.

4.3.1.4.1 ASSERTION SUB-ELEMENT

The *AssertionType* is used to define the security property that is to be certified by a Certification Model. It is a complex type as shown in Figure 11. According to this type, an assertion has a unique *ID* and is defined through a sequence of guaranteed terms (see sub-element *Guaranteed*). A guarantee term defines the conditions that must be monitored at runtime, in order to come to a conclusion on whether or not a security property is satisfied or violated.

The definition of guaranteed conditions is based on, and refers to, the specification of interfaces of the cloud services and mechanisms that need to be certified (the ToC). These

interfaces are specified by the elements *InterfaceDeclr*, as shown in Figure 11. They may also make use of one or more variables, which are defined by the *VariableDecl* elements. Where necessary, the definitions of such interfaces and variables need also to be specified as part of the *Assertion* element.

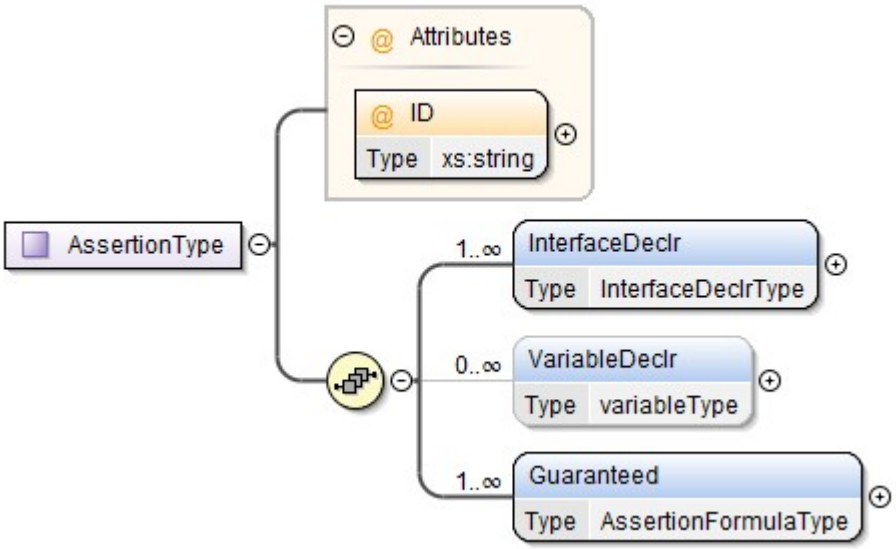


Figure 11 – AssertionType

4.3.1.4.2 INTERFACEDECL SUB-ELEMENT

The specification of an interface declaration (*InterfaceDecl*) sub-element, as presented in Figure 12, consists of:

- The sub-element *ID* that uniquely identifies the *InterfaceDeclr*. This *ID* is a mandatory sub-element.
- The sub-element *ProviderRef* that uniquely identifies the party that is obligated to provide the interface. The *ProviderRef* is also a mandatory sub-element.

- The sub-element *Interface* that defines the interface of the service that needs to be certified. This sub-element may be one of the following types:
  - An inline Interface Specification (*InterfaceSpec*),
  - An interface reference (*InterfaceRef*) to an externally located *InterfaceSpec*, identified by the UUID attribute *'InterfaceLocation'*, or
  - A resource type (*InterfaceResourceType*), denoting a resource as a service ("Resource-as-a-Service")

  Interface sub-elements are used in cases where the definition of a security property relates to operations that belong to the service. For example, the availability of a service may be required for some of the interfaces that the service offers, but not all of them. In such cases only the interfaces that the availability property will refer to, need to be specified.

- Zero or more *Endpoint* sub-elements. These elements specify the endpoints that implement the specific interface of the service that need to be certified, and at which the interface operations can be invoked. Each Endpoint element is specified by:
  - A compulsory *ID* of the endpoint (this *ID* must be unique within the scope of the enclosing *InterfaceDeclr* element),
  - A compulsory *Location* element of type *string*, which records the address of the endpoint, and
  - A *Protocol* element specifying the communication protocol required for invoking the interface operations at the specific endpoint. This can take a value to denote specific communication protocols, such as SOAP, REST, or SSH.

Endpoint sub-elements indicate where a service can be invoked. For the monitoring based certification process this information is useful, because it will determine where the monitoring events should be captured from, in order to acquire the required monitoring evidence to generate a monitoring based certificate.

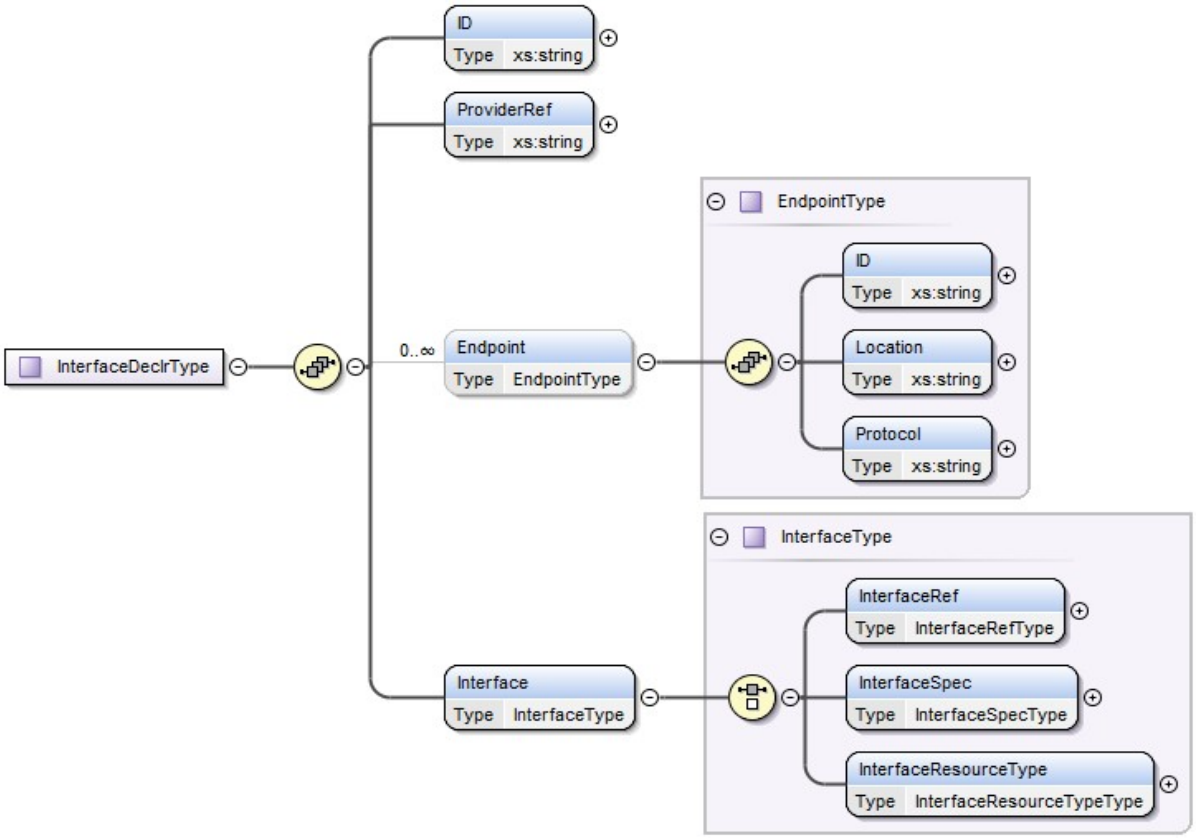


Figure 12 – Interface Declaration Type

The XML schema specification of the *InterfaceDeclrType* is given below.

```
<xs:complexType name="InterfaceDeclrType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="ID"
                type="xs:string"/>
    <xs:element maxOccurs="1" minOccurs="1" name="ProviderRef"
                type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="Endpoint"
                type="EndpointType"/>
    <xs:element maxOccurs="1" minOccurs="1" name="Interface"
                type="InterfaceType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="EndpointType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="ID"
```

```
                type="xs:string"/>
    <xs:element maxOccurs="1" minOccurs="1" name="Location"
                type="xs:string"/>
    <xs:element maxOccurs="1" minOccurs="1" name="Protocol"
                type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="InterfaceType">
  <xs:choice>
    <xs:element maxOccurs="1" minOccurs="1" name="InterfaceRef"
                type="InterfaceRefType"/>
    <xs:element maxOccurs="1" minOccurs="1" name="InterfaceSpec"
                type="InterfaceSpecType"/>
    <xs:element maxOccurs="1" minOccurs="1"
                name="InterfaceResourceType"
                type="InterfaceResourceTypeType"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="InterfaceRefType">
  <xs:sequence>
    <xs:element name="InterfaceLocation" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="InterfaceResourceTypeType">
  <xs:simpleContent>
    <xs:extension base="xs:string"> </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="InterfaceSpecType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="Name"
                type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="Extended"
                type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="Operation"
                type="operationType"/>
  </xs:sequence>
</xs:complexType>
```

4.3.1.4.3 VARIABLEDECL SUB-ELEMENT

The sub-element *VariableDeclr* is of type *variableType*, presented in Figure 13, which can be used to define variables, in order to express conditions within assertions, for the formal specification of security properties. As shown in the figure, the *variableType* has two attributes, which are the:

- *persistent* attribute that indicates whether the value of the variable is the same throughout all instances (like static variables in Java), and the
- *forMatching* attribute that distinguishes between internal and external variables (i.e. its value is false for internal variables).

Also, this type consists of the following sub-elements:

- *varName* that is of type *String* and signifies the name of the variable; and
- one of the following sub-elements:
  - *varType* of type *String* with a *value* element of type *String* or a value element of object type; or
  - an *array* element of type *arrayType* with elements that describe the array structure.

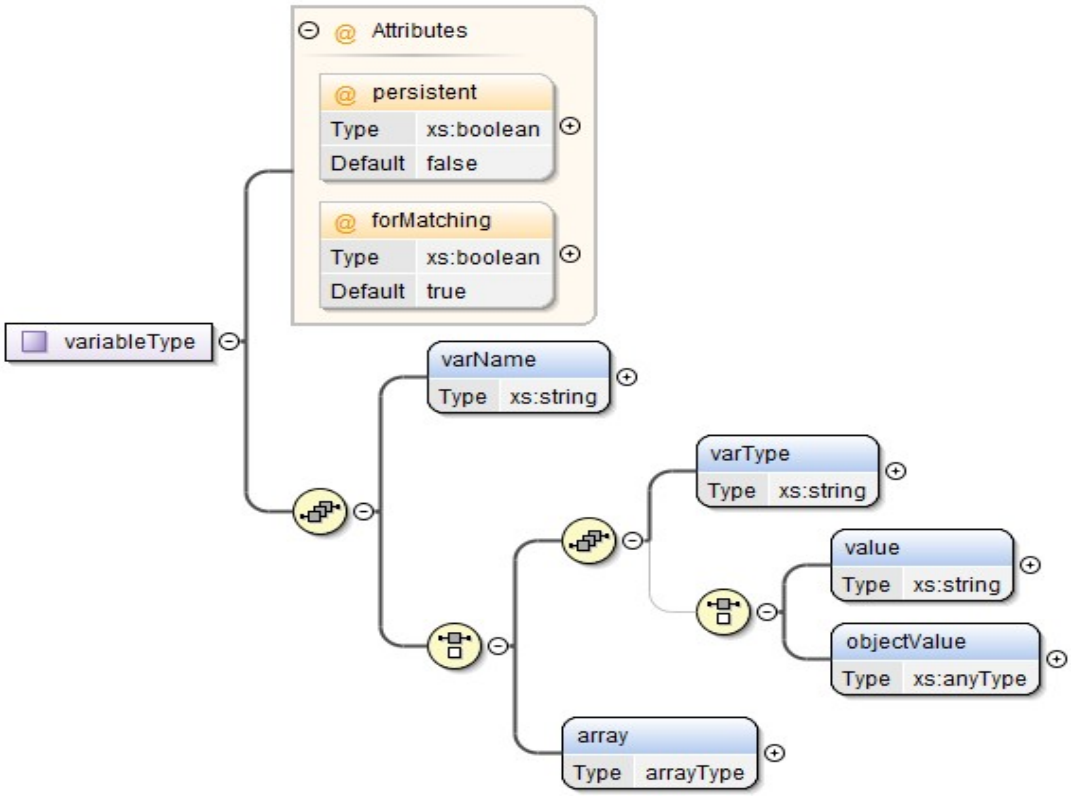


Figure 13 – Variable Type

The XML schema specification of the *variableType* is given below.

```
<xs:complexType name="variableType">
  <xs:sequence>
    <xs:element name="varName" type="xs:string"/>
    <xs:choice>
      <xs:sequence>
        <xs:element name="varType" type="xs:string"/>
        <xs:choice minOccurs="0">
          <xs:element name="value" type="xs:string"/>
          <xs:element name="objectValue" type="xs:anyType"/>
        </xs:choice>
      </xs:sequence>
      <xs:element name="array" type="arrayType"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute default="false" name="persistent" type="xs:boolean"/>
  <xs:attribute default="true" name="forMatching" type="xs:boolean"/>
</xs:complexType>
```

4.3.1.4.4 GUARANTEED SUB-ELEMENT

The sub-element *Guaranteed* is part of the *Assertion* element. Is of complex type *AssertionFormulaType*, as shown in Figure 14, and defines a guarantee that a certain state of affairs will hold. The *AssertionFormulaType* has two attributes:

- The attribute *ID* that is the unique id of the formula, and
- The attribute *type* that signifies whether the guarantee is
  - A *future* assertion, which is an assertion that needs to be checked against information that will arise after the occurrence of the events that will trigger the check of the assertion, or
  - A *past* assertion, which is an assertion that should be checked against information that exists at the time point when the events that trigger the check occur.

AssertionFormulaType also contains the following sub elements:

- A list of *quantification* elements that define the quantifiers for the variables and time variables, used to specify conditions of the assertion. Two types of quantification may exist for a variable:
  - The *existential* (exists), or
  - The *universal* (forall),
- An optional *precondition* element, which is of type *AssertionCondition* and determines the conditions under which the assertion should be checked (i.e., the conditions which if become true should trigger the checking of the assertion), and
- A *postcondition* element, which is of type *AssertionCondition*, and determines the conditions that are guaranteed to hold (i.e. should become true if the preconditions are true).

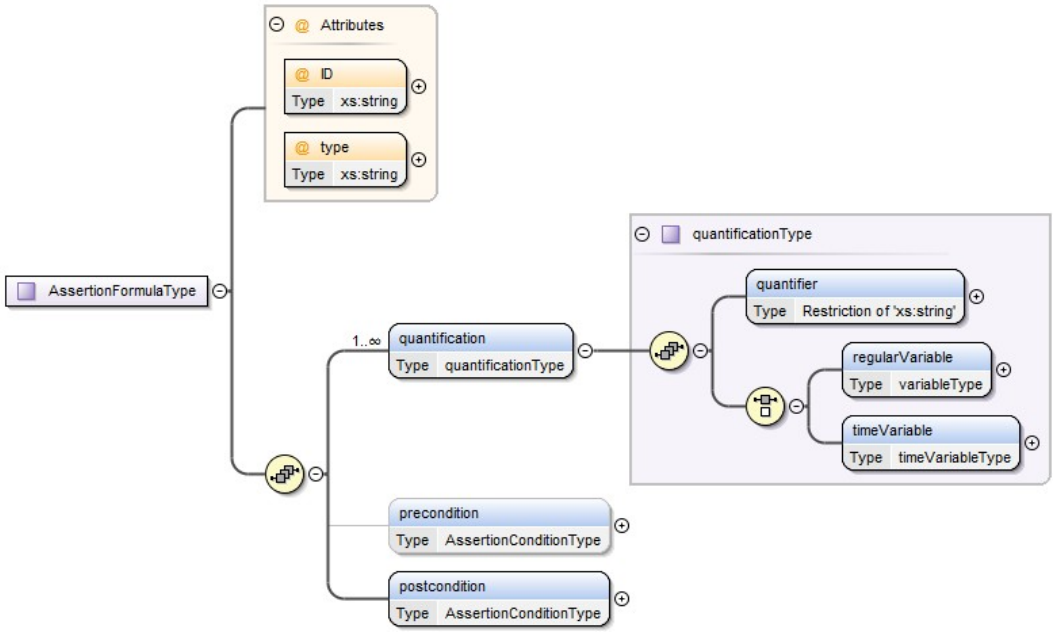


Figure 14 – AssertionFormulaType

The XML schema specification of the *AssertionFormulaType* is presented below.

```
<xs:complexType name="AssertionFormulaType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="1"
                name="quantification"
                type="quantificationType"/>
    <xs:element minOccurs="0" name="precondition"
                type="AssertionConditionType"/>
    <xs:element name="postcondition" type="AssertionConditionType"/>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:complexType>
```

The type *AssertionConditionType* enables the definition of atomic or complex logical conditions. This is enabled by the structure of this type, which is shown in Figure 15. More specifically, according to *AssertionConditionType*, an element of this type contains:

- An atomic condition of type *AssertionAtomicCondition*, and
- An optional sequence of a logical operator, which is an element of type *logicalOperationType*, which can be either:
  - An *AssertionCondition* element, or
  - A *timeCondition* element.

An *AssertionAtomicCondition* element has an attribute named *conditionID* that is the unique id of the condition (within the assertion that the condition belongs to), and can be of three different types:

- An *event* condition, which is of a type *eventConditionType*,
- A *state* condition, which is of type *stateConditionType*, or
- A *relational* condition, which is of type *relationalConditionType*.

Event conditions are conditions regarding the occurrence of events related to the ToC that the assertion, which includes the condition, refers to (e.g., the occurrence of an invocation (call) of an operation in one of the ToC's interfaces or a response to such a call).

A *state condition* is a condition regarding the state of the system that is being monitored at a given time point (e.g., a condition stating a certain user has already logged in to it or that the system is TPM enabled).

A *relational condition* is a condition regarding the value of a variable used in an assertion (e.g., a condition requiring a variable to have a certain value, or a condition requiring two variables to have the same value).

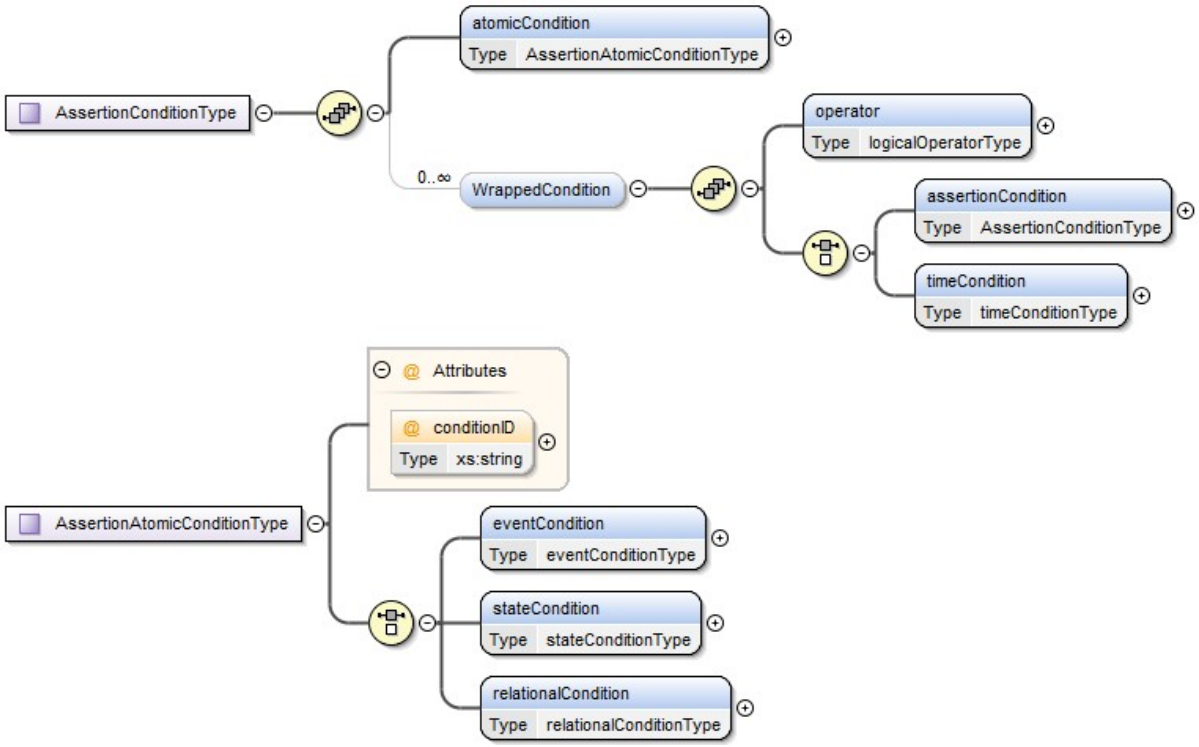


Figure 15 - Assertion Condition and Assertion Atomic Condition

The XML schema specification of the *AssertionConditionType* and *AssertionAtomicConditionType* is given below.

```
<xs:complexType name="AssertionConditionType">
  <xs:sequence>
    <xs:element name="atomicCondition"
                type="AssertionAtomicConditionType"/>
    <xs:element name="WrappedCondition" minOccurs="0"
                maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="operator" type="logicalOperatorType"/>
          <xs:choice>
            <xs:element name="assertionCondition"
                        type="AssertionConditionType"/>
            <xs:element name="timeCondition"
                        type="timeConditionType"/>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
```

```
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AssertionAtomicConditionType">
  <xs:choice>
    <xs:element name="eventCondition" type="eventConditionType"/>
    <xs:element name="stateCondition" type="stateConditionType"/>
    <xs:element name="relationalCondition"
                type="relationalConditionType"/>
  </xs:choice>
  <xs:attribute name="conditionID" type="xs:string"/>
</xs:complexType>
```

Figure 16 presents the event conditions, which are defined as elements of the type *eventConditionType*. An event condition is a condition regarding the occurrence of an event that can be of a type:

- *call* event that refers to a call of an operation,
- *reply* event that refers to a response to a call of an operation, or
- *execute* event that refers to an execution of an operation that must be invoked by the monitor itself.

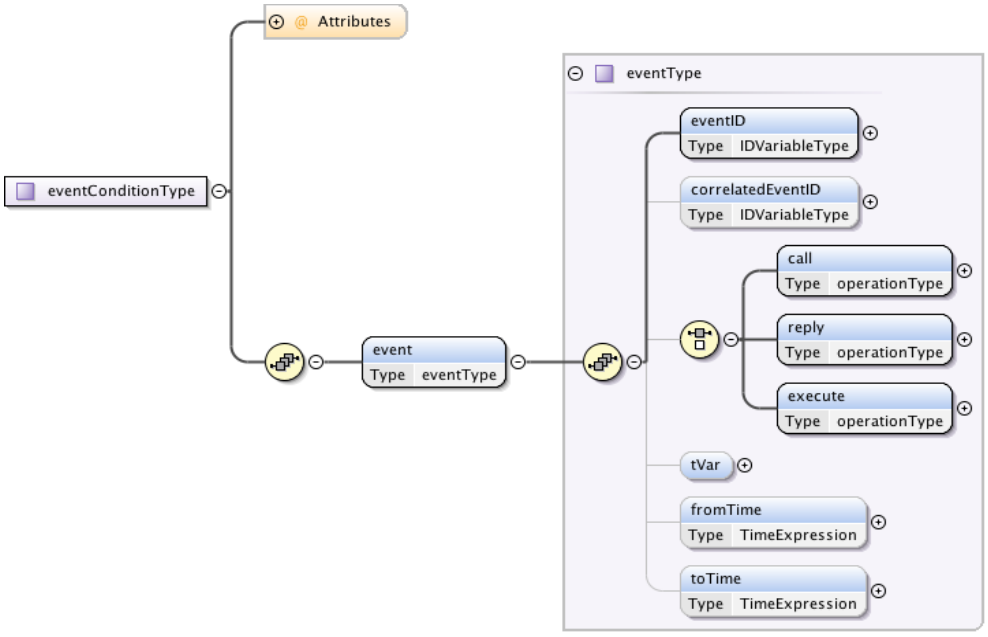


Figure 16 - Event Condition Type

The specification of the *eventConditionType* in XML schema is listed below.

```
<xs:complexType name="eventConditionType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="event"
                type="eventType"/>
  </xs:sequence>
  <xs:attribute default="false" name="negated" type="xs:boolean"/>
  <xs:attribute default="false" name="unconstrained"
                type="xs:boolean"/>
  <xs:attribute default="false" name="recordable" type="xs:boolean"/>
  <xs:attribute default="false" name="abducible" type="xs:boolean"/>
</xs:complexType>

<xs:complexType name="eventType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="eventID"
                type="IDVariableType"/>
    <xs:element maxOccurs="1" minOccurs="0" name="correlatedEventID"
                type="IDVariableType"/>
    <xs:choice maxOccurs="1" minOccurs="0">
      <xs:element name="call" type="operationType"/>
      <xs:element name="reply" type="operationType"/>
      <xs:element name="execute" type="operationType"/>
    </xs:choice>
    <xs:element name="tVar" minOccurs="0">
```

```
      <xs:complexType>
        <xs:choice>
          <xs:element name="timeVar" type="timeVariableType"/>
          <xs:element name="timePeriod" type="TimePeriodType"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="fromTime" type="TimeExpression" minOccurs="0"/>
    <xs:element name="toTime" type="TimeExpression" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

An event condition is defined by the following sub-elements:

- An *eventID* of the type *String* that identifies uniquely the event of the condition,
- An optional *correlatedEventID,* which refers to another event that may be related to this event (e.g., an event representing the response to a call of a ToC operation may be explicitly correlated with the event that represents the call),
- A choice of elements *call*, *reply* or *execute* that determines whether the event of the condition is a call, reply or execute event, respectively. All these three types of events are defined as elements of the type *operationType*. Elements of this type define the signature of the operation, which is called, replied to or executed. The definition of *operationType* is shown in Figure 17 and is explained below.
- An element *tVar* that defines the point in time at which the event is expected to occur. A *tVar* element can be defined as either
  - *timeVar* element, of type *timevariableType*, or
  - *timePeriod* element, of type *TimePeriodType*.

  A *timeVar* element is used when the event is expected to occur at a single instance of time and is of type *timevariableType*, which consists of:

  - A *varName*, of type *String*, for specifying the name of the variable,
  - A *varType*, of type *String*, which has fixed value *TimeVariable*, and
  - A *value* element, of type *String*, for specifying the value of the variable (i.e., the time instance at when the relevant event occurs).

A *timePeriod* element is defined by two attributes:

- A (time) *period*, and
- A (time period) *unit*.

A *tVar* is defined by a *timePeriod* element if the event that it is associated with is an execution event, such as an operation call, that must be executed periodically.

- A time range within which the event must occur. This element consists of two sub-elements to define the upper and the lower time boundary of the time range. These elements are:
  - The *fromTime* element that defines the lower boundary of the time range of an event, and
  - The *toTime* element that defines the upper boundary of the time range of an event.

  The elements *fromTime* and *toTime* are both of type *TimeExpression* (see Figure 18). A *TimeExpression* can be defined as a linear function over constants and/or time variables that have been introduced in the relevant assertion. To enable this, the definition of a *TimeExpression* element consists of:

  - A time element that is of type *timevariableType*; and
  - An optional expression starting by a time operator, i.e., *plusTime*, *minusTime*, *plus* or *minus*.

    The *plusTime* and *minusTime* operators are of type *timevariableType* and are used to introduce further time variables in the time expression. The operators *plus* and *minus* are both of *decimal* type and are used to introduce constants in the expression. This structure enables the definition of time expressions such as: t1 + t2, t1 + t2 + 3.0, t1 + 2.0[1].

[1] All constant decimal values in a time expression are assumed to express time in nanoseconds.

[2] The "[" and "]" denote a closed range at the lower and upper boundary respectively, and "(" and ")" denote an open range at the lower and upper boundary respectively.

[3] This assertion is specified in XML in Appendix A

[4] Atos Spain S.A is an international information technology services company that works with clients across the

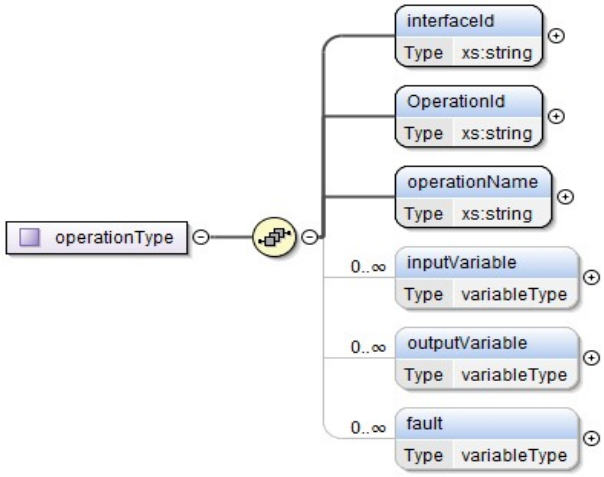


Figure 17 – Operation Type

The definition of the *operationType* as shown in Figure 17, consists of:

- An element called *interfaceId* that is of type *String* and denotes the interface that the operation belongs to,
- An element, called *operationId*, that is of type *String* and signifies the unique id of the operation within the interface,
- An element, called *operationName*, that is of type *String* and denotes the name of the operation,
- Zero or more *inputVariable* elements, which are of type *variableType* and define the input variables (parameters) of the operation,
- Zero or more *outputVariable* elements, which are of type *variableType* and define the output variables (parameters) of the operation, and
- Zero or more *fault* elements, which are of type *variableType* and define the fault variables of the operation.

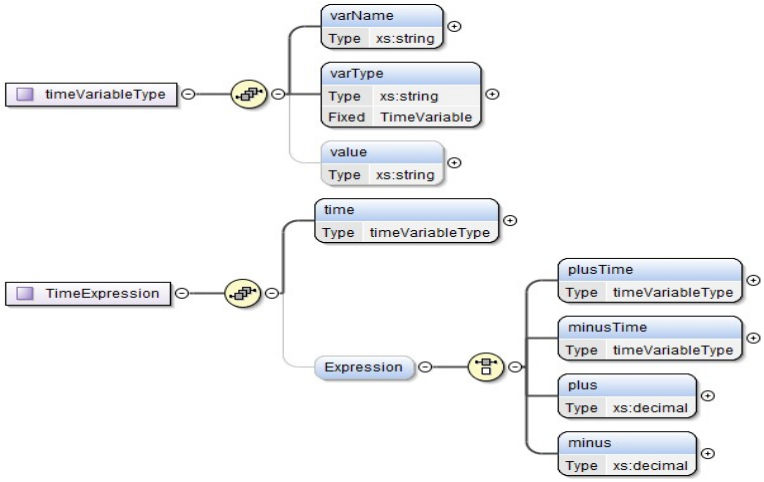


Figure 18 - Time variable Type and Time Expression Type

The XML schema for the *operationType*, *timeVariableType* and *TimeExpression* is given below.

```
<xs:complexType name="operationType">
  <xs:sequence>
    <xs:element name="interfaceId" type="xs:string"/>
    <xs:element name="OperationId" type="xs:string"/>
    <xs:element name="operationName" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0"
                name="inputVariable"
                type="variableType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0"
                name="outputVariable"
                type="variableType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="fault"
                type="variableType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="timeVariableType">
  <xs:sequence>
    <xs:element name="varName" type="xs:string"/>
    <xs:element fixed="TimeVariable" name="varType" type="xs:string"/>
    <xs:element minOccurs="0" name="value" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="TimeExpression">
  <xs:sequence>
    <xs:element name="time" type="timeVariableType"/>
    <xs:element name="Expression" minOccurs="0">
      <xs:complexType>
        <xs:choice>
          <xs:element name="plusTime" type="timeVariableType"/>
          <xs:element name="minusTime" type="timeVariableType"/>
          <xs:element name="plus" type="xs:decimal"/>
          <xs:element name="minus" type="xs:decimal"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

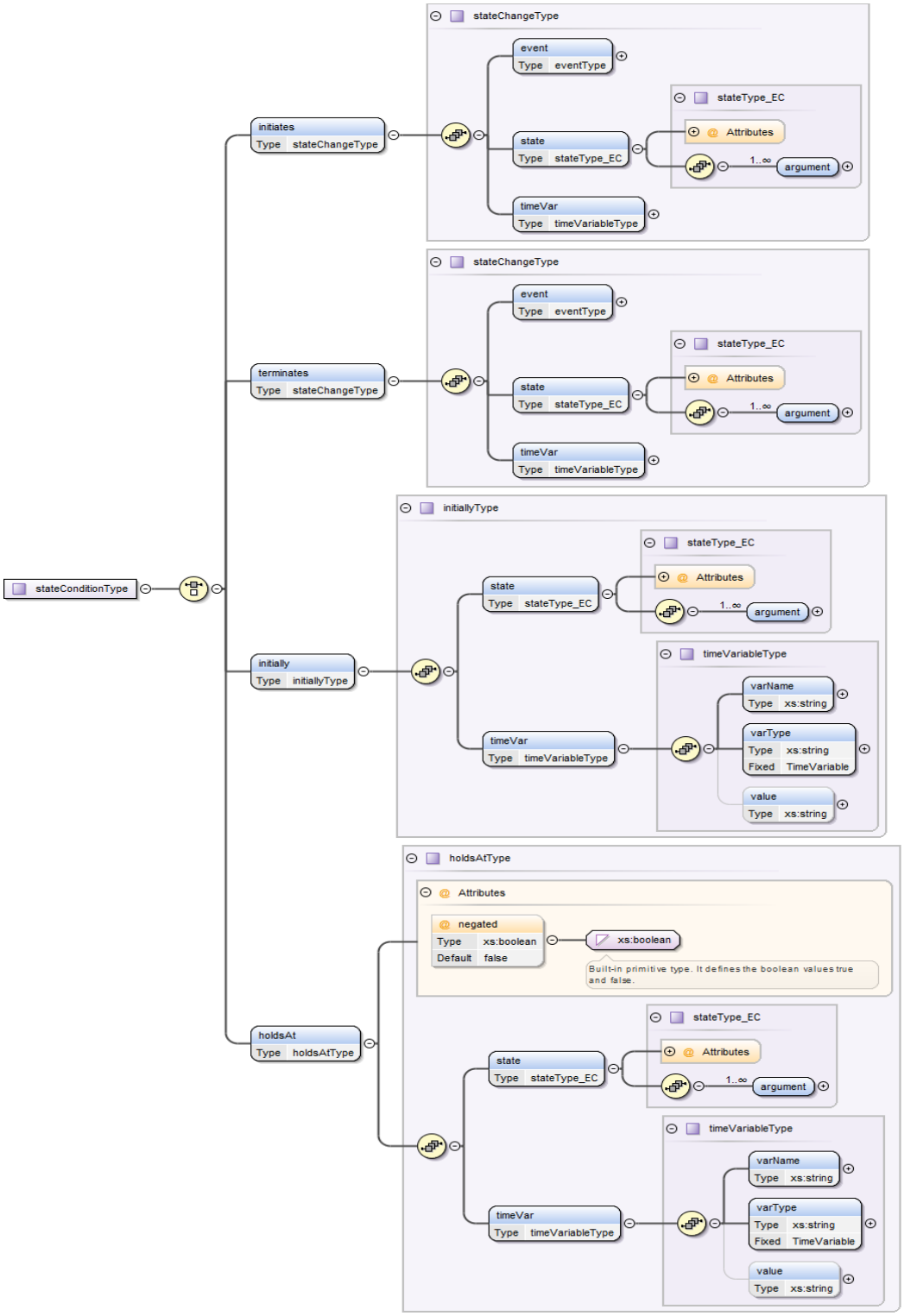


Figure 19 - State Condition Type

The second type of atomic conditions in an assertion, are the *stateConditions*. *stateConditions* refer to the state of the system that is being monitored in a particular instance of time. A *stateConditions* may, for example, be that a particular user u1 is successfully logged into a system with a role r1 at a particular instance of time t1. Conditions are expressed by n-ary relations of the form "relation-name" (arg1, …, argn). In line with Event Calculus, such relations can be set up at the beginning of the operation of a system or initiated by events that occur at specific time points during the operation of the system. They can also be terminated by other events. From the time that a state condition is initiated by an event and until the time that it is terminated by an event, the condition holds (i.e., it is assumed to be *True*). In the case of our previous example, the state condition expressing that the u1 has been logged in with role r1, would be expressed by the relation loggedIn(u1,r1). This condition would be initiated by a logging in event of u1 and would be terminated by a logging out event of u1.

In our assertion language, *stateConditions* can be specified as instances of the type *stateConditionType*. The structure of this type is shown in Figure 19. The *stateConditionType* supports the specification of elements expressing the initiation, termination and holding of state conditions. In particular, a *stateConditionType* element can be:

- An *initiates* element that is of type *initiatesType* and expresses the initialisation of some state by some event at some time point. The definition of elements of *stateChangeType* consists of:
  - An *event* element (i.e., an element of type *eventType*), which expresses the event which causes the initialisation of the state value;
  - A *state* element, which expresses a state of the system that is being monitored as a relation between a list of arguments that have specific values; and
  - An element *timeVar*, which expresses the time when the state element was initialised (*timeVar* is of complex type *timevariableType*),
- A *terminates* element that expresses the termination of some state value by some event. *terminates* elements are similar with *initiates* elements in expressing the effect of an event on the state of a system at a given time point. Thus, they are also defined as instances of the type *stateChangeType*,

- A *holdsAt* element that represents the system state which is true (i.e., holds) at specific time point. As shown in Figure 19, this element is defined as instance of the type *holdsAtType*. According to this type, its definition consists of the following elements:
  - A *state* element that represents the state value (see *initiates* element above) and
  - A *timeVar* that represents the time when the *state* is held (this element is of complex type *timevariableType*).

  Furthermore, this type has an attribute *negated*, of a type *Boolean*, that indicates if the *holdsAt* element is negated or not, and

- An *initially* element that represents a state value at the beginning of a monitored period of system operation. As shown in Figure 19, this element is defined as instance of the type *initiallyType*. According to this type, its definition consists of the following elements:
  - A *state* element that represents the state value (see *initiates* element above) and
  - A *timeVar* that represents the time when the state is held (this element is of complex type *timevariableType*).

The XML schema specification of the *stateConditions* is provided below.

```
<xs:complexType name="stateConditionType">
  <xs:choice>
    <xs:element name="initiates" type="initiatesType"/>
    <xs:element name="terminates" type="initiatesType"/>
    <xs:element name="initially" type="holdsAtType"/>
    <xs:element name="holdsAt" type="holdsAtType"/>
  </xs:choice>
</xs:complexType>
```

The third type of atomic conditions that may define an assertion, are the *relationalConditions*. These conditions enable the specification of transient conditions about the values of variables used in the assertions (i.e., a condition regarding the relation between values of two variables or a condition between a variable and a value). We call these conditions transient because they are checked at the time of the evaluation of an assertion but there is no permanent recording of their form and truth-value following the completion of the check of an assertion. An example of a

relational condition is a condition requiring that one of the input variables of an operation, which has been invoked by a call event, should have a particular value at the time of the invocation.

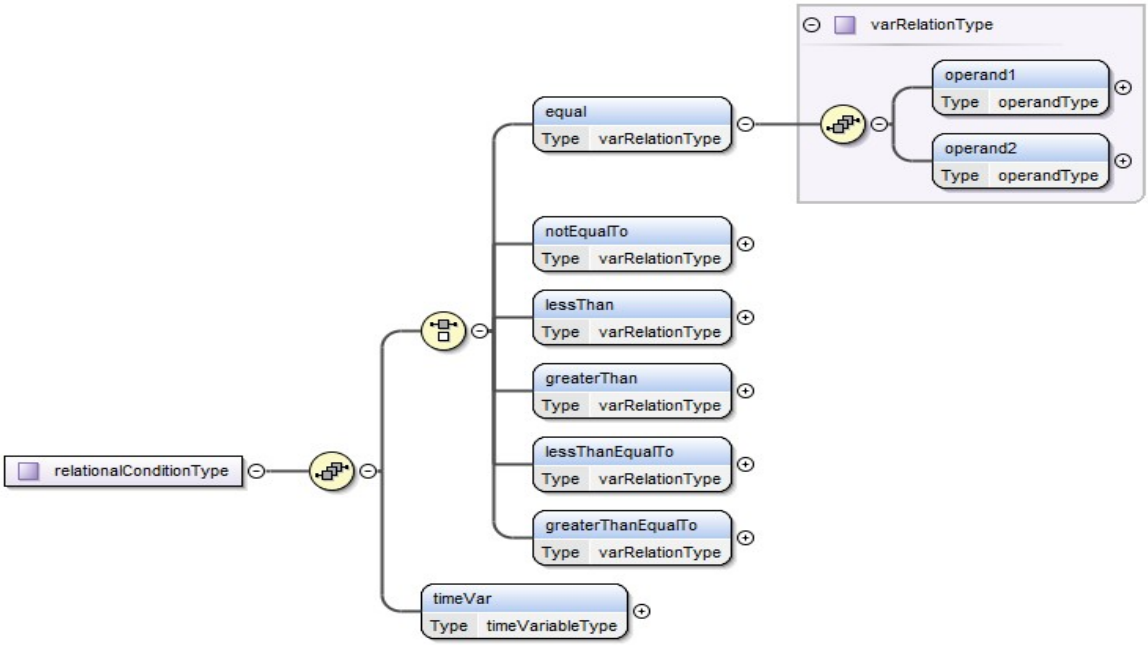


Figure 20 – RelationalConditionType

Figure 20 shows the structure of the type of relational conditions, *relationalConditionType*. As shown in the figure, *relationalConditionType* can be one of the following conditions:

- *equalto*, defining that the *operand1* should be equal to *operand2*,
- *notEqualTo*, defining that the *operand1* should not be equal to *operand2*,
- *lessThan*, defining that the *operand1* should be less than *operand2*,
- *greaterThan*, defining that the *operand1* should be greater than *operand2*,
- *lessThanEqualTo*, defining that the *operand1* should be less than or equal to *operand2*, and
- *greaterThanEqualTo*, defining that the *operand1* should be greater than or equal to *operand2*.

The above relational conditions are expressed by correspondingly named sub-elements of *relationalConditionType*. All these alternative sub-elements are instances of the complex type *varRelationType*, which defines the two elements involved in the relation, which are the *operand1* and the *operand2*. These elements are called *operands* and are of type *operandType*.

The XML schema specification of the *relationalConditionType* and the*varRelationType* is given below.

```
<xs:complexType name="relationalConditionType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="equal" type="varRelationType"/>
      <xs:element name="notEqualTo" type="varRelationType"/>
      <xs:element name="lessThan" type="varRelationType"/>
      <xs:element name="greaterThan" type="varRelationType"/>
      <xs:element name="lessThanEqualTo" type="varRelationType"/>
      <xs:element name="greaterThanEqualTo" type="varRelationType"/>
    </xs:choice>
    <xs:element name="timeVar" type="timeVariableType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="varRelationType">
  <xs:sequence>
    <xs:element name="operand1" type="operandType"/>
    <xs:element name="operand2" type="operandType"/>
  </xs:sequence>
</xs:complexType>
```

Finally, a relational condition element has also a *timeVar* element that expresses the time point at which the relational condition should hold. *timeVar* elements are expressed as instances of the type *timevariableType*, which was discussed above.

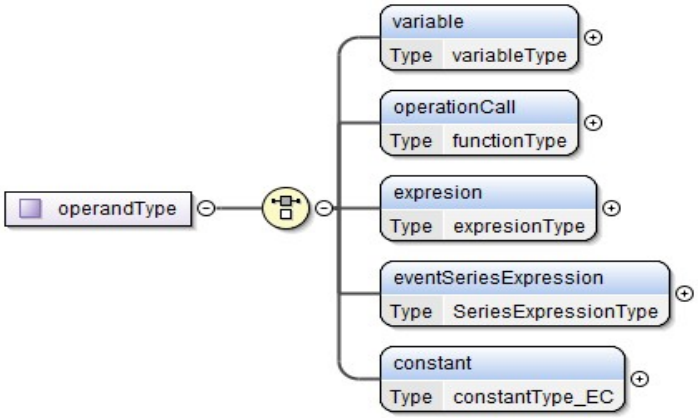


Figure 21 - Operand Type

The specification of the *OperandType* in XML schema is listed below.

```
<xs:complexType name="operandType">
  <xs:choice>
    <xs:element name="variable" type="variableType"/>
    <xs:element name="operationCall" type="functionType"/>
    <xs:element name="expresion" type="expresionType"/>
    <xs:element name="eventSeriesExpression"
                type="SeriesExpressionType"/>
    <xs:element name="constant" type="constantType_EC"/>
  </xs:choice>
</xs:complexType>
```

The definition of relation operands is according to their complex type *operandType*. As shown in Figure 21, an *operandType* element can be defined as:

- A *variable* element of type *variableType*.
- A *constant* element that denotes a constant value (these elements are of type *constantType_EC*).
- An *operationCall* element of type *functionType*. As shown in Figure 22, a *functionType* contains the following elements:
  - An element, called *name*, of type *String* that specifies the name of the function,
  - An element, called *partner*, of type *String* that signifies the partner service that will provide the function, and

- Zero or more *argument* elements, which may be of one of the following types: an *eventSeriesVariable*, a *variable* element, a *constant* element or a *function* element of *functionType*.

An *operationCall* element defines a function that is to be executed by the monitor or an external party, identified by the *partner* element of the call. It also defines the arguments for the specific function, which can be *variables* that can take different values during the monitoring process, *constants* or *event series*. When a *function* operand is encountered during the evaluation of a relational condition, the values of its arguments are established first, then the function is executed (this may involve calling an operation in an external party), and the result of the function replaces the function in the condition prior to its evaluation. If an exception arises during the execution of the function, the value of it becomes undefined and the relational condition becomes false.

- An *eventSeriesExpression* element of type *SeriesExpressionType*, which specifies the computation over a series of values of some arguments. As shown in Figure 23, *SeriesExpressionType* consists of the following elements:
  - An *eventSeriesCondition* element of type *AssertionConditionType* that signifies the conditions that produces the series values, and
  - A *computation* element of type *valueExpType*, which signifies the computation that should be performed, over the series values. As shown in Figure 23, *computation* element contains either an *execute* element of type *operationType* or a *function* element of type *functionType*.

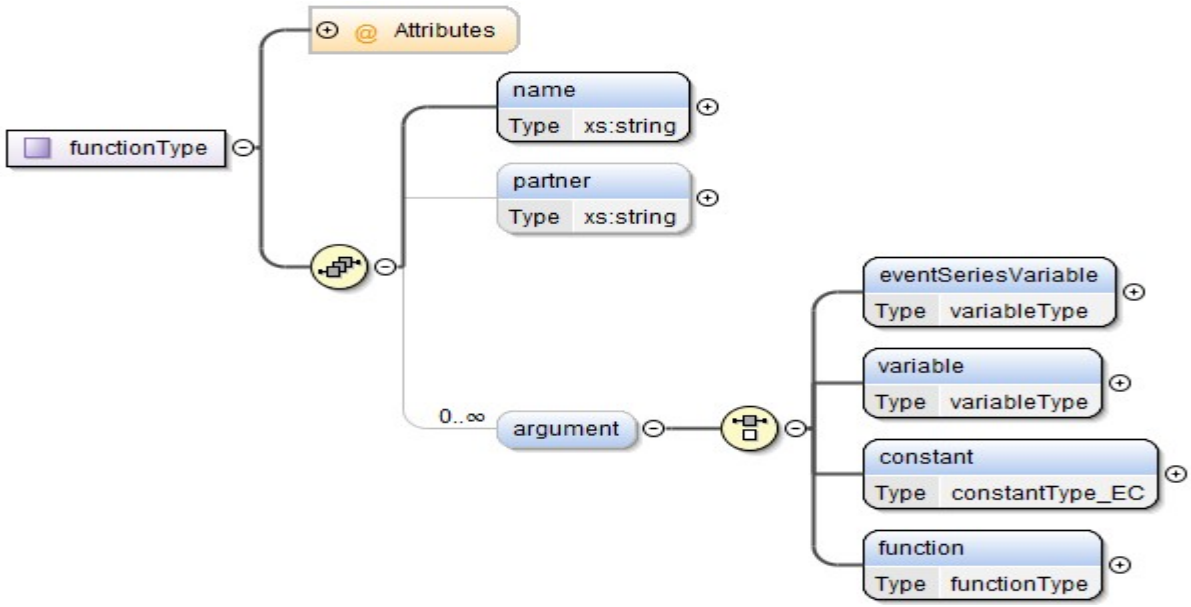


Figure 22 – FunctionType

The XML schema specification of the *functionType* is provided below.

```
<xs:complexType name="functionType">
  <xs:sequence>

<!-- operation restricted to a standard operation (i.e., min, max, avg, median,
mode, stdev, ...) -->
    <xs:element name="name" type="xs:string"/>
    <xs:element minOccurs="0" name="partner" type="xs:string"/>
    <xs:element name="argument" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:choice>

<!-- variable restricted to one of those in the event expression -->
          <xs:element name="eventSeriesVariable" type="variableType"/>
          <xs:element name="variable" type="variableType"/>
          <xs:element name="constant" type="constantType_EC"/>
          <xs:element name="function" type="functionType"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:string" use="required"/>
</xs:complexType>
```

The *event series* construct in the language has been introduced to support the succinct specification of properties that require the detection of complex event patterns occurring repeatedly within an event stream and, based on the results of this process, compute an aggregate value. Availability properties based on metrics like mean-time-between-failures or mean-time-to-repair are examples of such properties. Monitoring the mean-time-between-failures metric, for instance, would require monitoring a pattern of service consecutive failures and computing the average of the time difference between consecutive failures.

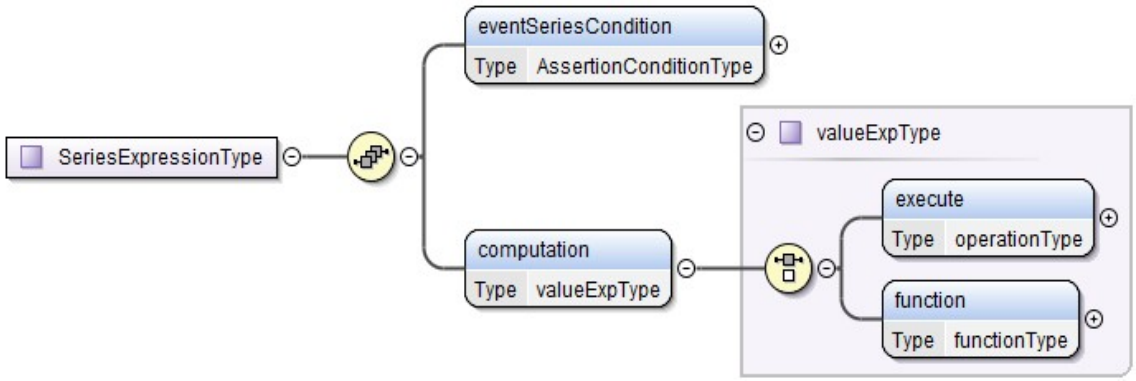


Figure 23 - SeriesExpressionType

The XML schema specification of the *SeriesExpressionType* and the *valueExpType* is given below.

```
<xs:complexType name="SeriesExpressionType">
  <xs:sequence>
    <xs:element name="eventSeriesCondition"
                type="AssertionConditionType"/>
    <xs:element name="computation" type="valueExpType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="valueExpType">
  <xs:choice>
    <xs:element name="execute" type="operationType"/>
    <xs:element name="function" type="functionType"/>
  </xs:choice>
</xs:complexType>
```

4.3.1.5 ASSESSMENTSCHEME ELEMENT

The element *AssessmentScheme* defines general conditions regarding the evidence that must be collected in order to be able to issue and maintain a certificate according to the particular certification model. These conditions are related to the i) sufficiency of evidence collection, such as the minimum period over which a target of certification must be monitored before a certificate for the particular property of it can be issued, ii) the expiration date of the instance, and iii) the absence of conflicting evidence regarding the security property to be certified, or iv) the absence of anomalous behaviour regarding the security property to be certified. These conditions must be satisfied, in addition to the guarantee states that are part of the assertion definition of the security property, for the certificate to be issued.

The definition of the type used for specifying *AssessmentScheme* elements is shown in Figure 24. As shown in the figure, the specification of an assessment scheme includes a sequence of:

- *evidence sufficiency* conditions (*EvidenceSufficiencyCondition* sub-element),
- *expiration* conditions (*ExpirationCondition* sub-element),
- *conflicts* (*Conflict* sub-element), and
- *anomalies* (*Anomalies* sub-element).

These sub elements of an assessment scheme are described below.

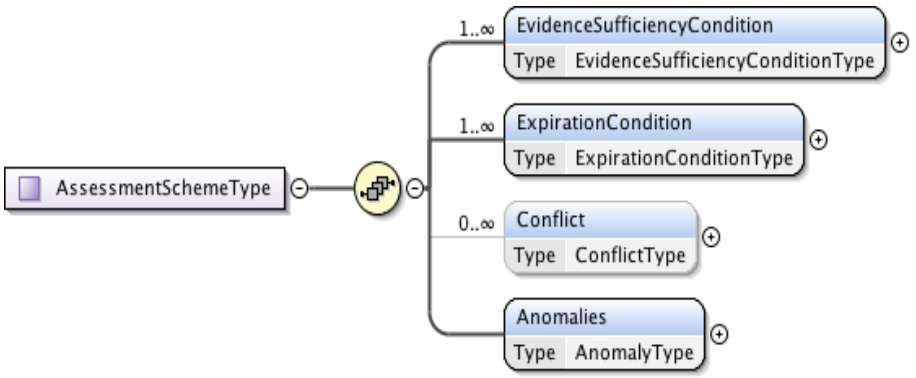


Figure 24 –AssessmentSchemeType

The XML schema specification of the *AssessmentSchemeType* is listed below.

```
<xs:complexType name="AssessmentSchemeType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded"
                name="EvidenceSufficiencyCondition"
                type="EvidenceSufficiencyConditionType"/>
    <xs:element minOccurs="1" maxOccurs="unbounded"
                name="ExpirationCondition"
                type="ExpirationConditionType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="Conflict"
                type="ConflictType"/>
    <xs:element name="Anomalies" type="AnomalyType"/>
  </xs:sequence>
</xs:complexType>
```

4.3.1.5.1 EVIDENCESUFFICIENCYCONDITION SUB-ELEMENT

The *EvidenceSufficiencyCondition* sub-element of the *AssessmentScheme* element, which is of a type *EvidenceSufficiencyConditionType*, is presented in Figure 26. This sub-element is used to describe conditions regarding the minimum extent of monitoring events or the minimum number and type of events that need to be monitored, before issuing a certificate.

It has a unique identifier (*Id*) and defines the conditions of the sufficiency conditions that must apply to evidence in order to issue a certificate. These conditions can be of three different types,

which are: i) the *MonitoringPeriodCondition*, ii) the *MonitoringEventsCondition*, or iii) the *ExpectedSystemOperationModelCondition*.

A) MonitoringPeriodCondition

A condition of this type can be used to define the period that a ToC should be monitored before a certificate can be issued. Such conditions can be specified according to the schema shown Figure 26. It should be noted that the security property to be monitored is determined by the assertions defined as part of the security property element of the model. The monitoring period defines only for how long these assertions should be monitored, before the evidence is deemed sufficient and a certificate can be issued.

B) MonitoringEventsCondition

A condition of this type can be used to define the minimum number of monitoring events that should be gathered before a certificate can be issued. Such conditions can be specified according to the schema shown in Figure 26. Similarly to the monitoring period element, the monitoring events condition determines the minimum number of events that must have been considered to issue a certificate.

C) ExpectedSystemOperationModelCondition

This is an optional condition that can be used to define an expected operation model of ToC (*ETOCB*). If such a model is defined, then the gathered evidence will be deemed sufficient for issuing a certificate only if the actual operation of ToC does not deviate from this model. Conditions of this kind are defined through a probabilistic state transition model of the behaviour of ToC and external actors interacting with it. This probabilistic state transition model describes the probabilities of occurrence of the events

that ToC should be expected to receive and the probabilities of the responses that it should be expected to produce.

More specifically, this element is a model that is specified as a deterministic automaton with expected relative event frequencies of the form:

```
ETOCB = <States, Events, sinit, PTrans, FinalStates>
```

In the ETOCB specification:

- *States* is the (finite) set of TOC states that are critical for the monitoring process;
- *Events* is the set of all possible events the TOC may produce that are of interest to certification;
- *sinit* is the initial TOC state;
- *PTrans* is a finite set of labelled transitions between two states; and
- *FinalStates* is the set of states where the certification automaton terminates. PTrans includes elements of the form (os, ds, e, R(lpr, upr)) where
    - *os* is the origin state of the transition,
    - *ds* is the destination state of the transition,
    - *e* is the signature of the event triggering the transition, and
    - *R(lpr,upr)* is the range of the expected relative frequency of undertaking this transition whilst the system is in *os*. *R(lpr, upr)* can be: (lpr, upr), [lpr, upr), (lpr, upr] or [lpr, upr]. The ETOCB model must satisfy some constraints. In particular: i) e must be an element of Events, i.e., an event denoting the invocation (or the response produced following an invocation) of an operation in the provided interface of TOC; ii) the boundaries lpr, upr should satisfy the conditions: $0 \leq lpr, upr \leq 1$, and $lpr \leq upr$; and iii) ETOCB must be a deterministic model.

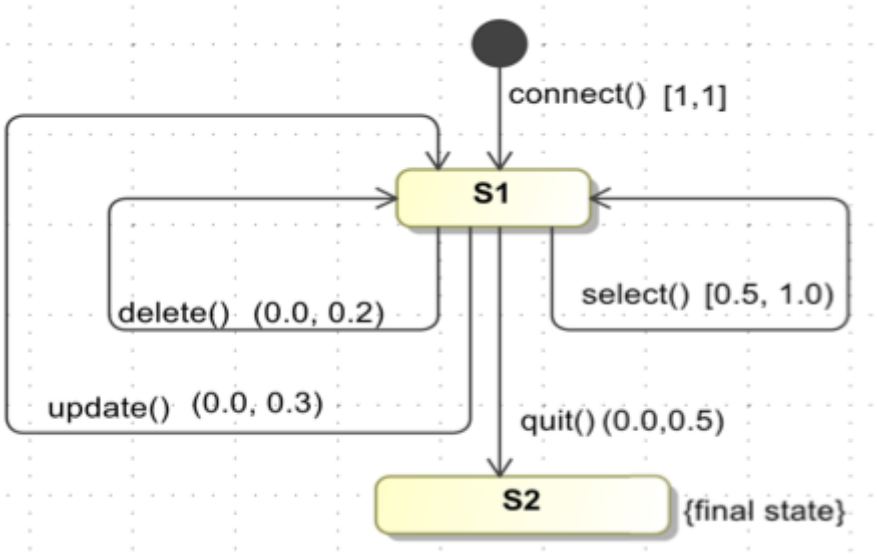


Figure 25 – ExpectedSystemOperationModel

ETOCB defines events that should be seen at different states during the operation of ToC (i.e., executed operations of the ToC) for the monitoring evidence to be sufficient. For certifying MySQL server, for example, this evidence should include executions of select, update, delete, and quit MySQL commands with specific frequencies. ETOCB is not required to be a complete model of TOC's behaviour; it only needs to define the states and events of importance for the property to be certified.

Figure 25 above gives an example of the ETOCB model for a relational DB server. This model expresses a view about the typical range of the server usage that should be taken into account in the certification of the server. According to it:

- The first interaction with the ToC should be a connect call to it, as stated by the event of transition from *InitialState* to *S1*, since a connection to the server should be established before any other query occurs. Also, according to the frequency range of this transition (i.e., [1,1]), connect calls should be the only initial event in any monitoring event trace, for the trace to be considered valid for the purposes of certification.

- Once a connection to the server is established, interactions with it may be requests for the execution of select(), update(), delete() or quit() operations (i.e., SQL queries) with expected frequency ranges [0.5, 1.0), (0.0, 0.3), (0.0, 0.2), and (0.0, 0.5), respectively, as indicated by the relevant transitions from *S1* to *S1* and *S2*. These expected frequency ranges require that data retrieval events (select() queries) will constitute at least half of the interactions with the server but data update() and delete() queries should also be seen. The model also expresses that:
    - It will be sufficient for certification purposes to see an event trace with update queries up to below 30% and delete queries up to below 20% of all interactions, and
    - Whilst at *S1*, the user may decide to quit(), according to the transition from *S1* to *S2*). Also, the lpr of the latter transition (i.e., lpr > 0) reflects that an event trace must always end with a quit() request for it to be a valid event trace for certification.

This type of element, since it defines a state transition model, it is of a type *StateTransitionModelType*, which is further described in Section 4.3.1.10.

The XML schema representation of the *EvidenceSufficiencyConditionType* is shown in Figure 26 below.

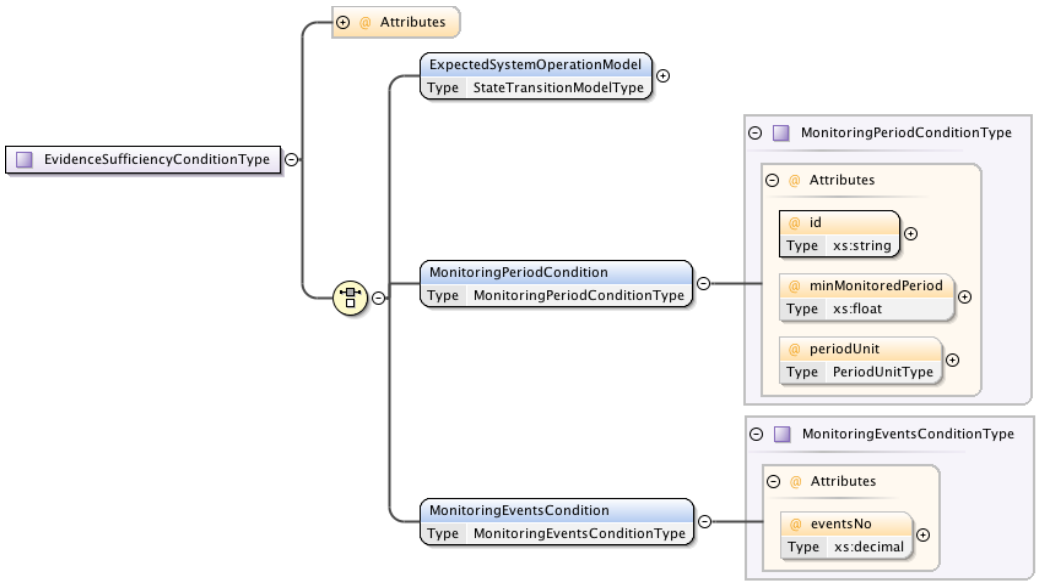


Figure 26 – EvidenceSufficiencyConditionType

The XML schema specification of the *EvidenceSufficiencyConditionType* is presented below.

```
<xs:complexType name="EvidenceSufficiencyConditionType">
  <xs:choice>
    <xs:element name="ExpectedSystemOperationModel"
                type="StateTransitionModelType"/>
    <xs:element name="MonitoringPeriodCondition"
                type="MonitoringPeriodConditionType"/>
    <xs:element name="MonitoringEventsCondition"
                type="MonitoringEventsConditionType"/>
  </xs:choice>
  <xs:attribute name="Id" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="MonitoringPeriodConditionType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="minMonitoredPeriod" type="xs:float"/>
  <xs:attribute name="periodUnit" type="PeriodUnitType"/>
</xs:complexType>
<xs:complexType name="MonitoringEventsConditionType">
  <xs:attribute name="eventsNo" type="xs:decimal" use="optional"/>
</xs:complexType>
```

4.3.1.5.2 EXPIRATIONCONDITION SUB-ELEMENT

An expiration condition defines when an issued certificate, which has been generated according to the given certification model, should expire and a new one could be issued by considering further evidence. This condition is expressed by an element of the certification model schema, called *ExpirationCondition*, which is of type *ExpirationConditionType*. The specification of *ExpirationCondition* elements in the XML schema is shown graphically in Figure 27.

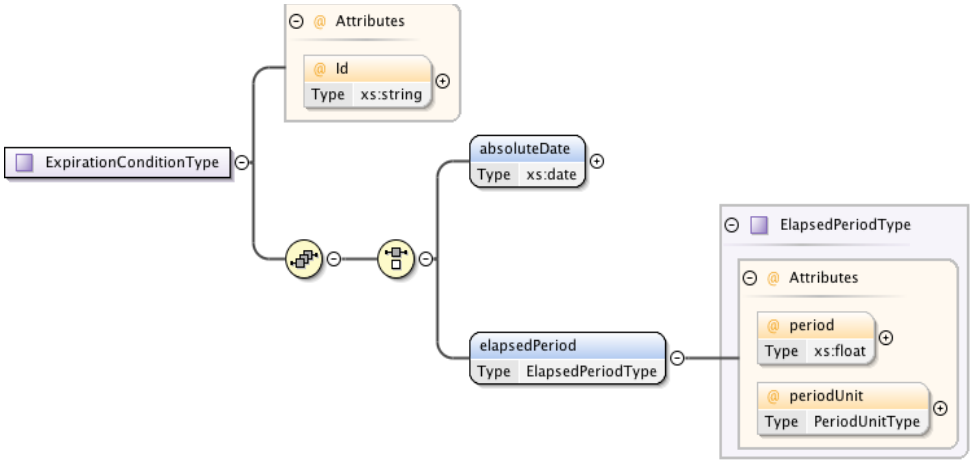


Figure 27 – Expiration Condition Type

The XML schema for *ExpirationConditionType* is listed below:

```
<xs:complexType name="ExpirationConditionType">
     <xs:sequence>
          <xs:choice>
               <xs:element name="absoluteDate" type="xs:date"/>
               <xs:element name="elapsedPeriod" type="ElapsedPeriodType"/>
          </xs:choice>
     </xs:sequence>
     <xs:attribute name="Id" type="xs:string"/>
</xs:complexType>

<xs:complexType name="ElapsedPeriodType">
     <xs:attribute name="period" type="xs:float"/>
     <xs:attribute name="periodUnit" type="PeriodUnitType"/>
</xs:complexType>
```

According to the above schema, an expiration condition has a unique identifier (*Id*) and a choice of two different ways to define the expiration date within it:

- As an *absoluteDate*, which is an element of type *date*, or
- As an *elapsedPeriod*, which is an element of type *ElapsedPeriodType*. An *ElapsedPeriod* element can be used when a certificate needs to expire at the end of a specific period of time from the date that it was issued.

  An *ElapsedPeriod* element expresses this by defining a period of time, as the number of time units that should elapse following the creation of the certificate, by defining two attributes,
  - the *period*, which of a type *float*, and
  - the *period unit*, which is of a type *PeriodUnitType* and can take values such as days, months, years etc..

4.3.1.5.3 CONFLICT SUB-ELEMENT

In a certification model conflicts define circumstances that may affect the state of a monitoring based certificate and the actions that should be taken in order to resolve it. More specifically, a conflict designates an assessment of the security property associated with the model for a sub period of the time specified in the certification model, which gives a different result from the assessment of the same property according to the assertion specified in the model. Thus, conflicts aim to capture cases where a given security property would not be satisfied if it were to be assessed over different monitoring aggregation periods.

Consider, for example, the case where a certification model used to certify the availability of a cloud storage service, within which the assessment of availability is based on average measures of the service availability taken over periods of one month. The availability of the service may, for instance, be above 99% if assessed on a monthly basis by certification model whose security property refers to this period of assessment, but it may be below this threshold if shorter or longer assessment intervals are considered. A conflict in this case would arise if the availability of the service may have been found to be less than 99%. Such discrepancies do not necessarily de-

validate certificates that lead to their suspension or revocation, but may need to be audited by the certification authority that has signed or will sign the certificate, before confirming the validity of an existing certificate or allowing a certificate to be issued.

In a CM conflicts are defined by alternative assessment periods for the security property. A conflict is defined by a *conflict* element in the certification model schema and is of the type *ConflictType*. The definition of *ConflictType* is shown in Figure 28.

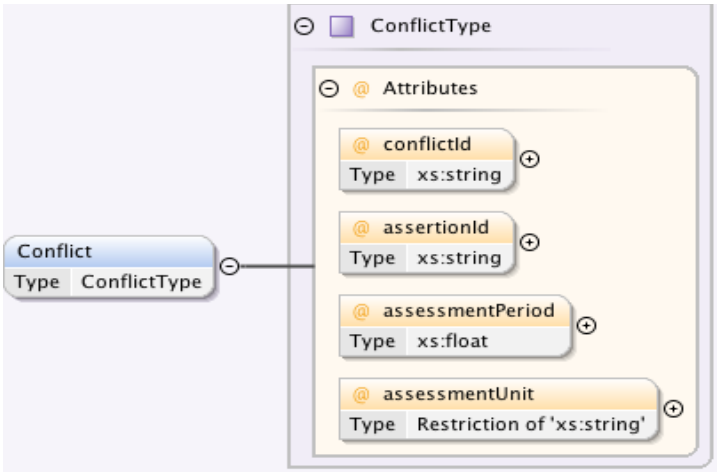


Figure 28 – Conflict Type

The specification of the *ConflictType* in the XML schema is shown below:

```
<xs:complexType name="ConflictType">
  <xs:attribute name="conflictId" type="xs:string"/>
  <xs:attribute name="assertionId" type="xs:string"/>
  <xs:attribute name="assessmentPeriod" type="xs:float"/>
  <xs:attribute name="assessmentUnit">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="nanoseconds"/>
        <xs:enumeration value="milliseconds"/>
        <xs:enumeration value="seconds"/>
        <xs:enumeration value="minutes"/>
        <xs:enumeration value="hours"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

According to this type, a *conflict* element is defined by the following four attributes:

- The *conflictId*, which is a unique identifier of the conflict element,
- The *assertionId* that identifies the assertion that the conflict element refers to, and
- The *assessmentPeriod* that determines the length of the period in time over which the assessment, whose purpose is to detect conflicts, should be conducted.
- *assessmentUnit* – this attribute determines the time unit in which the conflict assessment period is expressed

4.3.1.5.4 ANOMALIES SUB-ELEMENT

Certification models may also need to monitor and gather runtime evidence about i) potential attacks on ToCs, ii) other suspicious behaviour, or iii) operational conditions related to the security property, which despite not having caused any violation of the security property of the model so far, may lead to a violation of this property in the future. In the context of monitoring based certification models, we refer to these three possible cases as *anomalies*. Some *anomalies* may affect the status of certificates, i.e., they might lead to the suspension or revocation of a certificate. Other anomalies may only be used for auditing purposes.

The definition of the *anomalies* that should be monitored as part of a certification model, should be based on an analysis of whether potential attacks, or the ways in which the behaviour of different external actors that interact with ToC and the overall operating conditions of the interaction between ToC and these actors, may affect the satisfaction of the given security property by the ToC.

To specify the monitoring of anomalies we have introduced the following element, called *Anomalies*, as part of the assessment scheme of the monitoring based certification models. The type of this element is *AnomalyType*.

The structure of this type is shown in Figure 29 and the part of the XML schema that defines it is shown below:

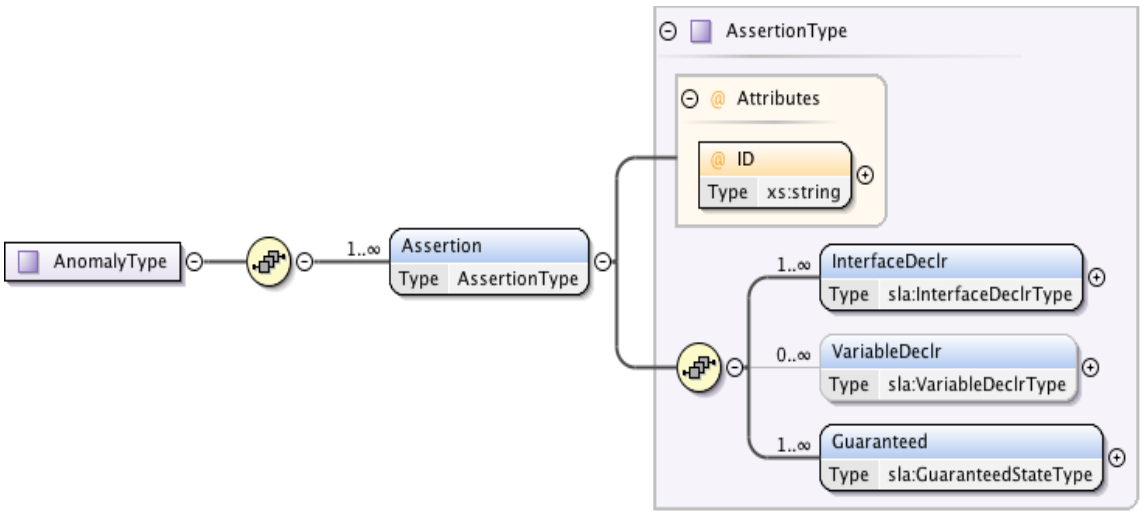


Figure 29 – AnomalyType

```
<xs:complexType name="AnomalyType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="Assertion"
                type="AssertionType"/>
  </xs:sequence>
</xs:complexType>
```

As shown in the above XML schema specification, an *anomaly* element is specified by one or more *assertions* that are to be monitored. These assertions are specified as elements of *AssertionType*, i.e., as normal monitoring conditions that should be checked during the acquisition of evidence for a monitoring based certificate.

The difference, however, from *assertion* elements specified as part of the *security property* element to be certified, is that the violation of the *assertion* elements of the *security property* element would typically either prevent the issuing of a certificate or lead to the revocation of an issued certificate. The *assertion* elements of the *anomaly* elements on the other hand, are used to gather monitoring evidence indirectly, which would typically need to be audited by the certification authority, which issues the certificates of the particular type, before any further action is taken. The way to treat detected anomalies should be specified in the life cycle model of the relevant certification model.

4.3.1.6 VALIDITYTESTS ELEMENT

A certification model may, in addition to the assessment scheme, define some extra validity tests as preconditions before issuing a certificate of a given type. Validity tests are optional extra preconditions that might need to be fulfilled before issuing a certificate of a given type, in addition to the assessment scheme's conditions. These tests may relate i) to conditions regarding the cloud infrastructure, in which the service to be certified is deployed (e.g., requiring that the cloud offers full isolation of virtual machines), ii) to the adherence of other services' protocol that the service to be certified may depend on (e.g., requiring that a storage service, which is used by a SaaS service, implements correctly a proof-of-retrievability protocol), or iii) to conditions regarding the monitoring infrastructure itself (e.g., requiring the integrity of the transmission of monitoring events and results inside the infrastructure and to external clients of it).

Such conditions are specified by the element *validityTests* in the certification model schema, as shown in Figure 30. The specification of validity tests can be based on expressing conditions regarding Trusted Computing based certificates or other types of certificates regarding ToC and its operational context, which can confirm the adherence of such entities to required conditions. For example, a validity test might be that the monitoring components, which have been used to gather the evidence underpinning a certificate, have integrity and have remained the same throughput the whole monitoring period. Thus, we expect that validity tests can be expressed as logical conditions over such certificates and their contents.

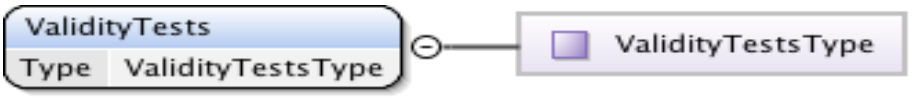


Figure 30 – Validity Tests Type

4.3.1.7 MONITORINGCONFIGURATIONS ELEMENT

In Figure 31 the *MonitoringConfigurations* element is presented, which is used to specify the list of the monitoring configurations that have been used to collect the evidence for generating certificates of this type.

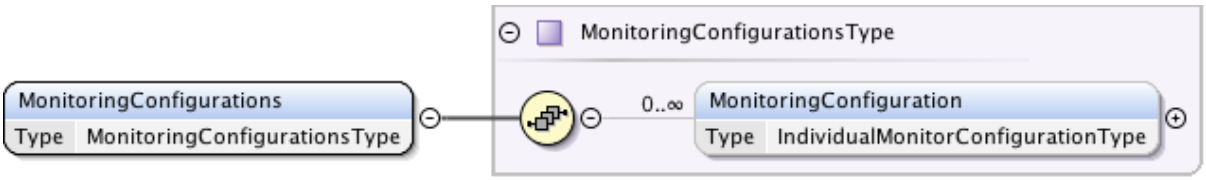


Figure 31 – Monitoring Configurations Type

As shown in Figure 32, each monitoring configuration includes:

- A unique Identifier (*Id*) as an attribute,
- A list of *Components* of the monitoring environment,
  These components can be of two types:
  - *sensors*, which are components capable of capturing and transmitting primitive monitoring events, and
  - *reasoners*, which are components capable of analysing events and checking whether monitoring conditions are satisfied, also known as *monitors*.

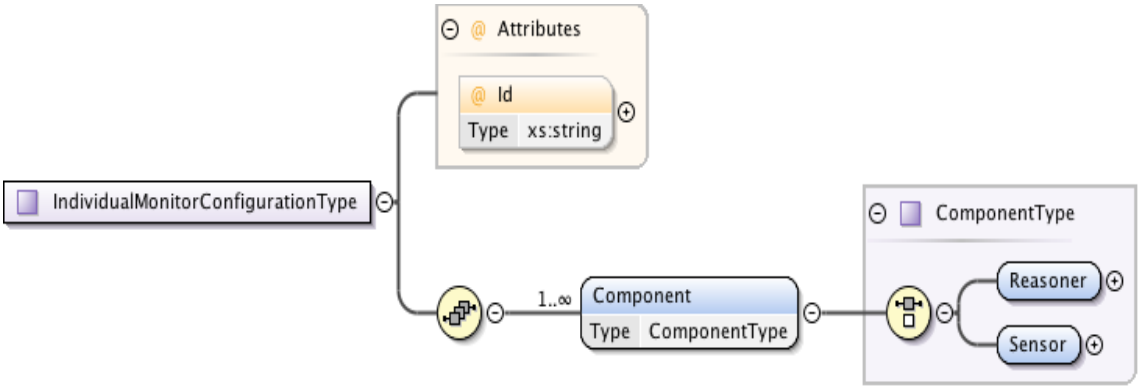


Figure 32 – Individual Monitor Configuration Type

The XML schema for monitoring configuration type is shown below.

```
<xs:complexType name="IndividualMonitorConfigurationType">
   <xs:sequence>
      <xs:element maxOccurs="unbounded" name="Component"
                  type="ComponentType"/>
      <xs:element name="ConcreteProperty" type="ec:formulasType"/>
   </xs:sequence>
   <xs:attribute name="Id" type="xs:string"/>
</xs:complexType>
<xs:complexType name="ComponentType">
   <xs:sequence>
      <xs:element name="EndPoint" type="xs:string"/>
   </xs:sequence>
   <xs:attribute default="REASONER" name="type">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="SENSOR"/>
            <xs:enumeration value="REASONER"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

4.3.1.8 EVIDENCEAGGREGATION ELEMENT

In Figure 33 is presented the *EvidenceAggregation* element, which defines how often should the monitoring evidence being checked, in order to update the evidence in the generated certificate, with new aggregated evidence.

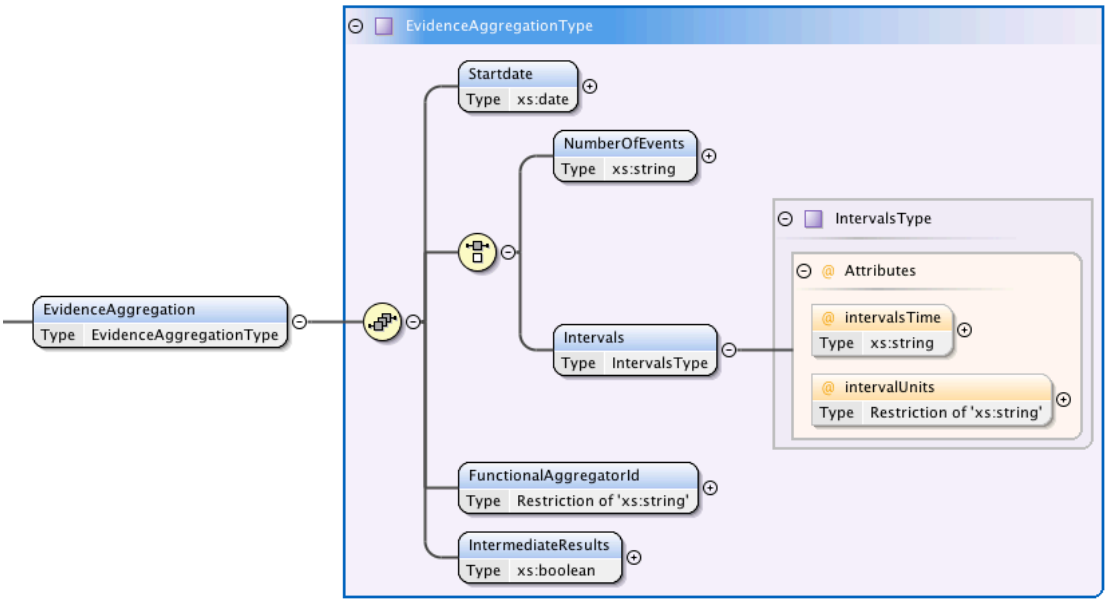


Figure 33 – Evidence Aggregation Type

As shown in the figure, this element consists of the following elements:

- The *StartDate* of the first aggregation,
- A choice of either:
  - A *NumberOfEvents*, which defines the number of the primitive monitoring events that should be aggregated, or
  - An *Interval* element, which consists of two attributes:

  - The *intervalsTime*, which specifies how often should the evidence be aggregated, and
  - The *intervalUnit*, which declares the unit used to specify the interval time.

- The *FunctionalAggregatorId*, which defines what type of aggregation should done in the events, and
- The *IntermediateResults* element, which is an optional Boolean element that could be used to specify if there is a need to aggregate evidence between two predefined aggregation periods, to check the validity of the certificate.

The XML schema specification of the *EvidenceAggregationType* is given below.

```
<xs:complexType name="EvidenceAggregationType">
  <xs:sequence>
    <xs:element name="AggregatedResultsInfo">
      <xs:complexType>
        <xs:attribute name="Startdate" type="xs:date"/>
        <xs:attribute name="Timestamp" type="xs:date"/>
        <xs:attribute name="NumberOfEvents" type="xs:string"/>
        <xs:attribute name="intervalsTime" type="xs:float"/>
        <xs:attribute name="intervalUnit">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="nanoseconds"/>
              <xs:enumeration value="milliseconds"/>
              <xs:enumeration value="seconds"/>
              <xs:enumeration value="minutes"/>
              <xs:enumeration value="hours"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="EventSummary">
      <xs:complexType>
        <xs:attribute name="NumberOfViolations" type="xs:string"
                      use="required"/>
        <xs:attribute name="NumberOfSatisfactions" type="xs:string"
                      use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="AggregatedValue" type="xs:string"/>
    <xs:element minOccurs="0" name="FunctionalAggregatorId">
      <xs:simpleType>
        <xs:restriction base="xs:string">
```

```
          <xs:enumeration value="Max"/>
          <xs:enumeration value="Min"/>
          <xs:enumeration value="Average"/>
          <xs:enumeration value="Standard Deviation"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="IntermediateResults" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

4.3.1.9 LIFECYCLEMODEL ELEMENT

The life cycle model (LCM) element in a certification model defines the process by which certificates can be generated and managed (e.g., monitored, issued, suspended, revoked). LCM is a compulsory element of a certification model as it enables a certification authority to specify with full precision the certification process, by defining the different states of certificates that can be generated by the certification model and which events should change it. During the operation of the framework, the LCM is used to monitor on-going certification processes, determine the state at which they are (e.g., collecting monitoring evidence, checking validity conditions prior to issuing a certificate) and, depending on it, update the state of the certificate that may be generated by the process.

A life cycle model (LCM) is defined as a state transition model of the form

LCM = <Sinit, States, Trans>

In an LCM, i) States is the finite set of states of it (a state may be an atomic state or a composite state specified by another embedded LCM); ii) Sinit is the initial state of the process; and iii) Trans is a finite set of transitions between two states.

Trans includes elements of the form (si, sj, e, g, a) where:

- *si* is the origin state of the transition;
- *sj* is the destination state of the transition;

- *e* is the signature of the event triggering the transition;
- *g* is guard condition that must be satisfied for the transition to take place; and
- *a* is a set of actions that should be executed if the transition takes place.

In an LCM, *e* must be an element of the provided interface of the framework (e.g., the operation enabling the notification of monitoring events, the operation to be executed if the user of the framework wishes to suspend or revoke a certificate).

An example of an LCM is shown in Figure 34. The LCM in the figure has an initial state called Activated and the states *InsufficientEvidence*, Pre-Issued, Issued, and Revoked. It also has two composite states: Continuous Monitoring and Issuing.

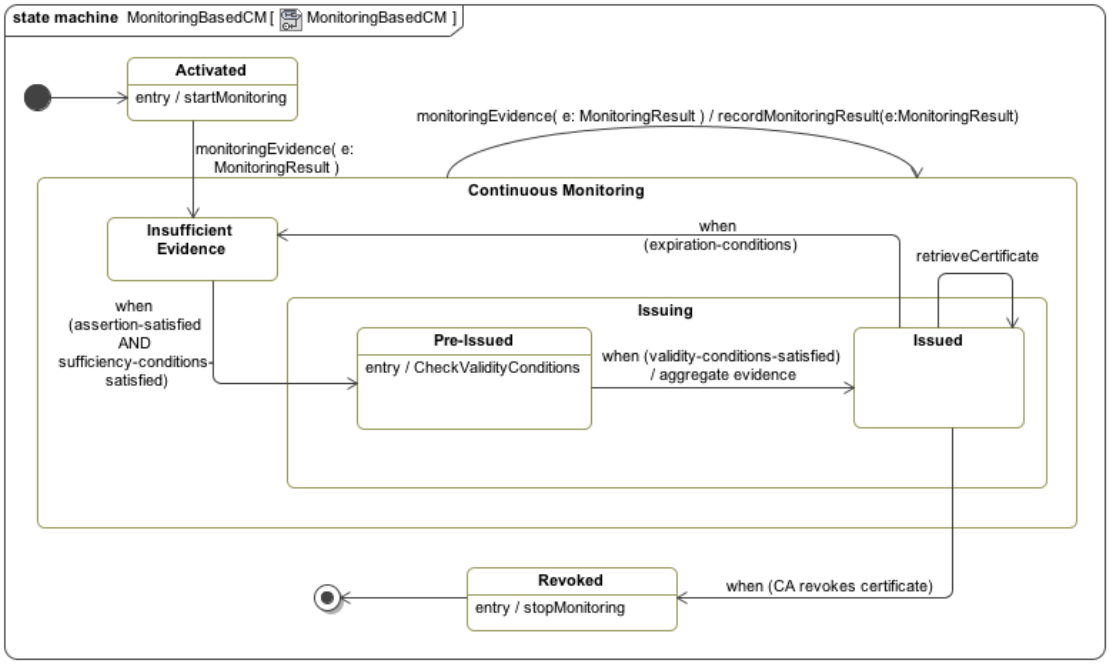


Figure 34 - UML diagram of Life Cycle Model

According to the model, after a certificate is activated, it moves to the *InsufficientEvidence* state, at which the monitoring evidence that is relevant to it starts getting accumulated. When the

accumulated evidence becomes sufficient according to the *EvidenceSufficiencyConditions* specified in the CM, and if there have been no violations of the monitoring rule that defines the security property (i.e., the security property of the CM is satisfied), the certificate moves to the state Pre-Issued. At this state, the certification infrastructure will check if the extra validity conditions for the certificate type (if any) are satisfied and, if they are, the certificate will move to the state Issued. In this state, any interested party with appropriate authority can retrieve the issued certificate from the framework. Whilst a certificate is at the Issuing state, monitoring continues and if a violation of the monitoring rule of the CM is detected, the certificate moves to the Revoked state at which it will no longer be valid and available.

Thus, the *LifeCycleModel* element defines all possible states that a certificate could take, from the time that it will be generated until it will cease to exist, and all the transitions between the different states, as well as their conditions, which should be references of the predefined conditions of the *AssessmentScheme* element of certification model. In Figure 35 the XML schema of this element is presented.

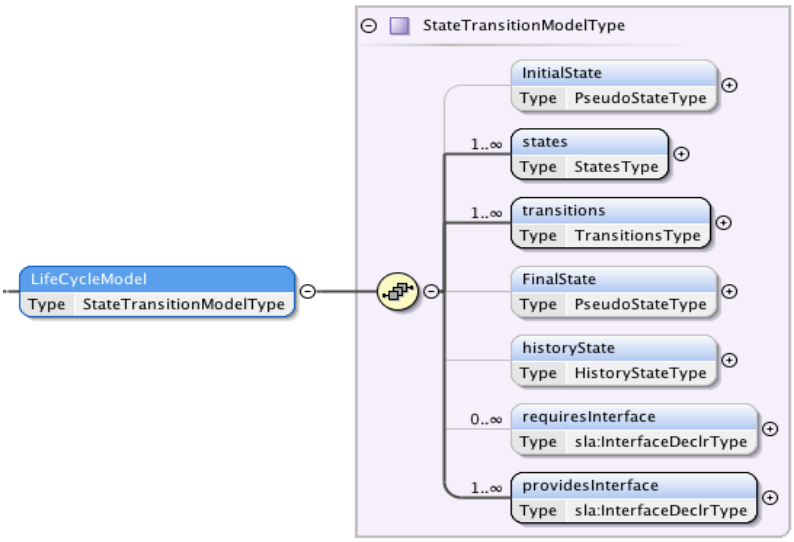


Figure 35 – LifeCycleModel Element

4.3.1.10 STATETRANSITIONMODELTYPE

The *LifeCycleModel* and the *ExpectedSystemOperationModelCondition* elements define state transitions models, thus they both are of a type *StateTransitionModelType*. Figure 36 shows a graphical representation of the structure of the XML schema that is used to specify state transition models. According to this type, a state transition model consists of:

- An optional *InitialState*,
- An optional *FinalState,*
- An optional *HistoryState,*
- A sequence of atomic or composite *states*,
- The *transitions* between states,
- One or more *provided interfaces*, which are sets of operations (or else interfaces) that are realised by the entity whose behaviour is described by the model, and
- One or more *required interfaces*, which are sets of operations that the entity, whose behaviour is described by the model, expects other external interacting entities to have

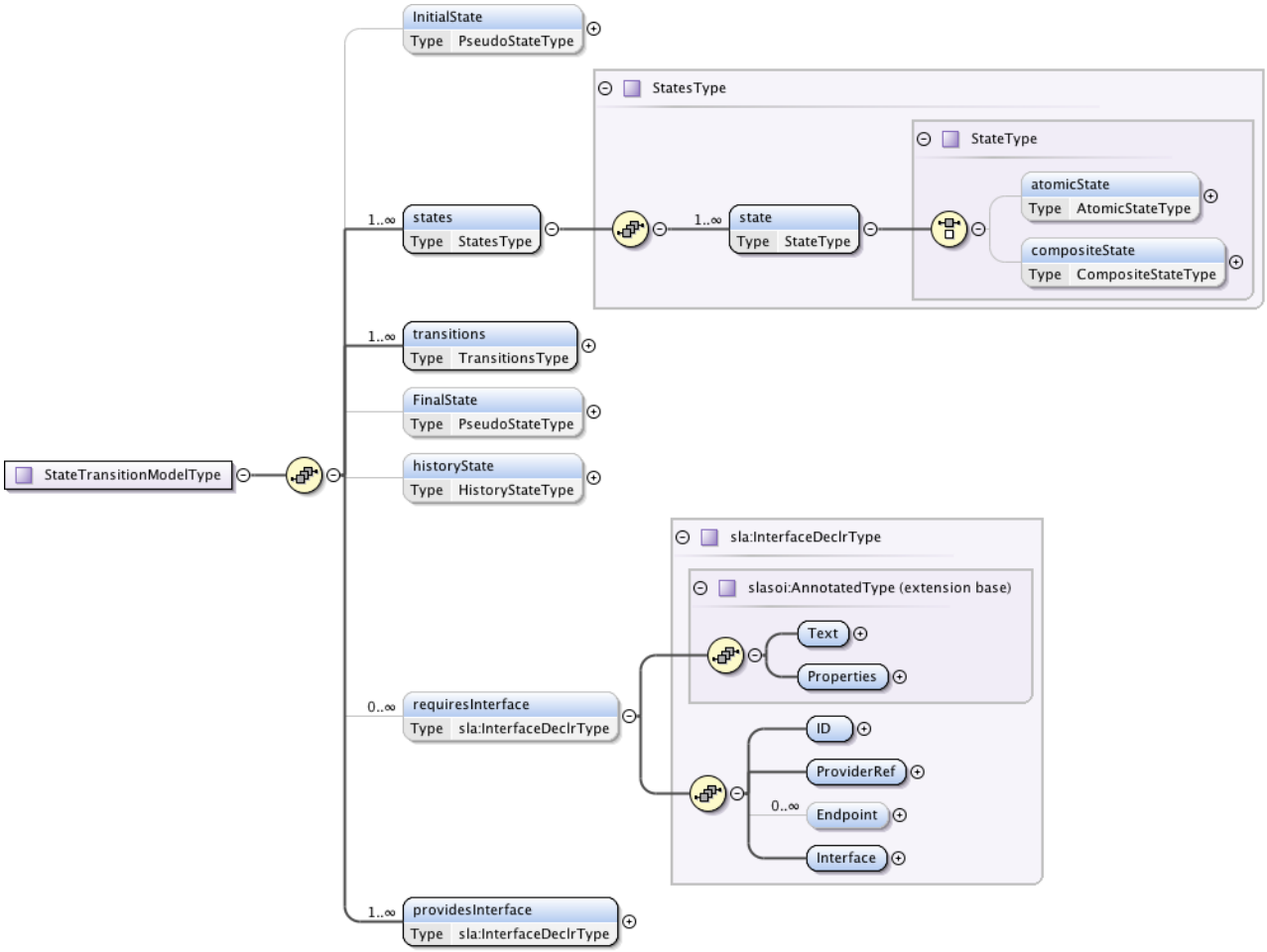


Figure 36 – State Transition Model Type

The XML schema specification of the *StateTransitionModelType* is listed below.

```
<xs:complexType name="StateTransitionModelType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="InitialState"
                type="PseudoStateType"/>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="states"
                type="StatesType"/>
    <xs:element minOccurs="1" maxOccurs="unbounded"
                name="transitions"
```

```
                type="TransitionsType"/>
    <xs:element minOccurs="0" maxOccurs="1" name="FinalState"
                type="PseudoStateType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
                name="historyState"
                type="HistoryStateType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
                name="requiresInterface"
                type="InterfaceDeclrType"/>
    <!-- interfaces that the LC model provides for (a) realisation of actions
and/or (b) call events that may force transitions -->

<xs:element minOccurs="0" maxOccurs="unbounded"
                name="providesInterface"
                type="InterfaceDeclrType"/>
    <!-- interfaces that it requires of external parties e.g., monitor's
interface that will be used by actions starting and ending the monitoring
process -->
  </xs:sequence>
</xs:complexType>

<xs:complexType name="StatesType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="state"
                type="StateType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="StateType">
  <xs:choice>
    <xs:element minOccurs="0" maxOccurs="1" name="atomicState"
                type="AtomicStateType"/>
    <xs:element minOccurs="0" maxOccurs="1" name="compositeState"
                type="CompositeStateType"/>
  </xs:choice>
</xs:complexType>
```

4.3.1.10.1 STATES SUB-ELEMENT

The *InitialState* and *FinalState* elements are pseudo states, which are used to designate the initial state of the model and the final state of the model respectively. Both these elements are specified as instances of the element type *PseudoStateType*, as shown in Figure 37.

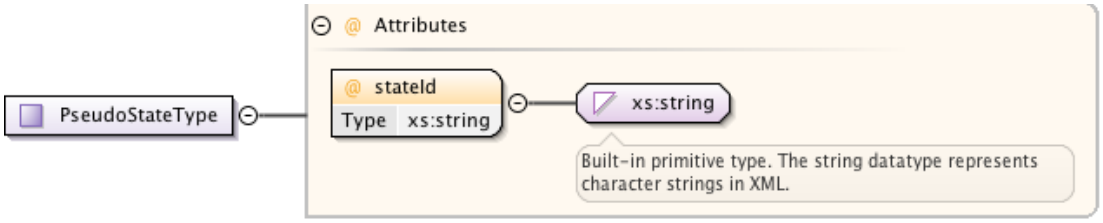


Figure 37 – Pseudo State Type

Pseudo states have only one attribute called *stateId* that uniquely identifies them within a state transition model.

A *historyState* element can be used in the cases where a state transition model is embedded within another state transition model, or more precisely a composite state of it (see composite states below). Historic state elements are used to keep a record of the last active atomic state within the composite state, before a transition was triggered to move the state of the relevant entity to a state outside the embedded model, where the historic state belongs. Historic state elements are described as instances of the element type *HistoricStateType*, which is defined in Figure 38 below.

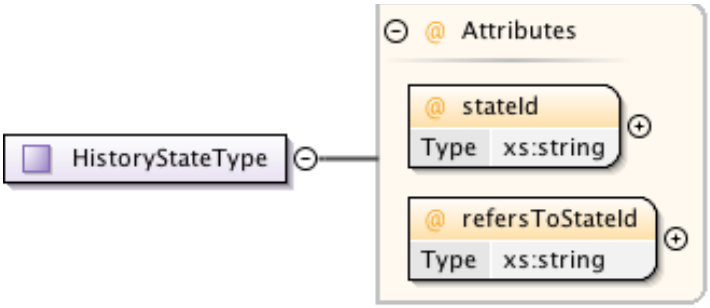


Figure 38 – History State Type

Apart from the *initial*, *final* and *historic* states, state transition models have also normal *states*. At least one of such states must be present in a model. These define periods in the life of the entity, whose behaviour is expressed by the model and over which the entity, that the state

transition model is associated with, waits for external events that may make it move from the particular state and/or take some actions. Normal states can be:

- *atomic* sates, or
- *composite* states.

Atomic states are specified by elements, which are instances of the type *AtomicStateType*. This type is shown in Figure 39. As shown in the figure an atomic state element is described by the following attributes:

- A *stateId,* which uniquely identifies the state within a state transition model,
- A *name*, which provides the chosen name of the state, and
- A *description*, which can be used to provide a description of the intended meaning of the state.

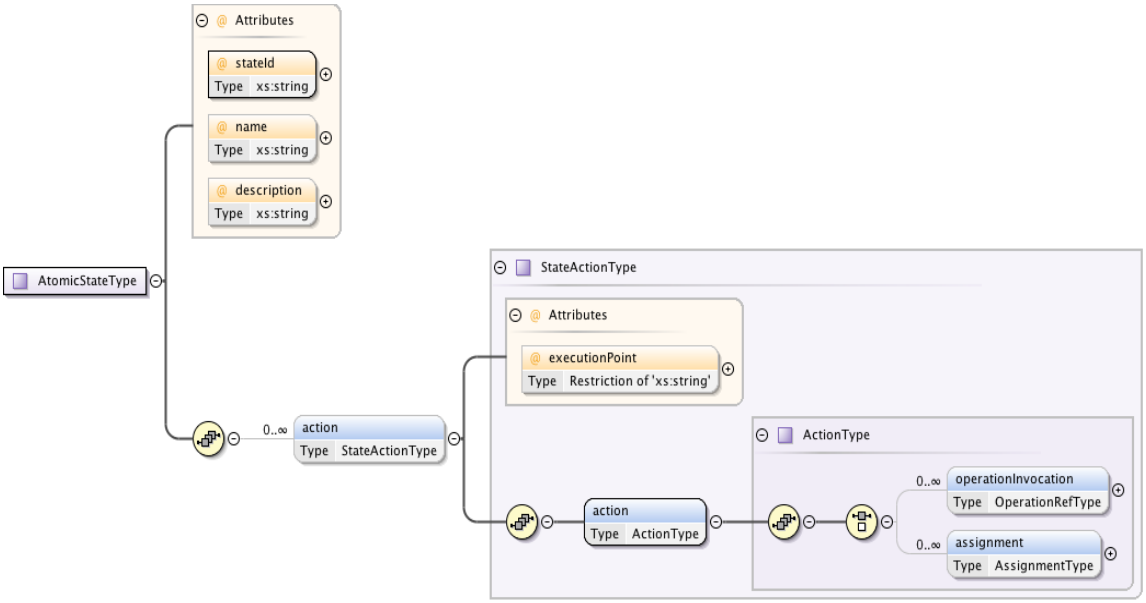


Figure 39 – AtomicStateType

The XML schema representation of the *AtomicStateType* is presented below.

```
<xs:complexType name="AtomicStateType">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="action"
                type="StateActionType"/>
  </xs:sequence>
  <xs:attribute name="stateId" type="xs:string" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="description" type="xs:string"/>
</xs:complexType>
```

Atomic states may also be associated with zero or more *actions* that must be executed when the entity is in them (for example update some internal variables, store etc.). *Actions* are described as elements of the type *ActionType* in the certification model specification schema, as shown in Figure 39. More specifically, an *action* has an attribute, called *executionPoint*, which defines whether the action is executed when the entity enters or exits the state (the fixed values "*onEntry*" and "*onExit*" define which of these two cases applies for an action, respectively). *Actions* can be of two kinds:

- They may require the execution of operations in the entity associated with the state transition model (i.e., an operation that belongs to one of the provided interfaces of the state transition model) or the invocation of an operation in an external entity (i.e., an operation that belongs to one of the required interfaces of the state transition model). Such actions are specified as *operationInvocation* elements.
- They may require the *assignment* of a value to a variable of the entity of the model. Such actions are specified as assignment elements.

The operationInvocation actions are specified according to the type *OperationRefType*. The graphical representation of this type is shown in Figure 40.

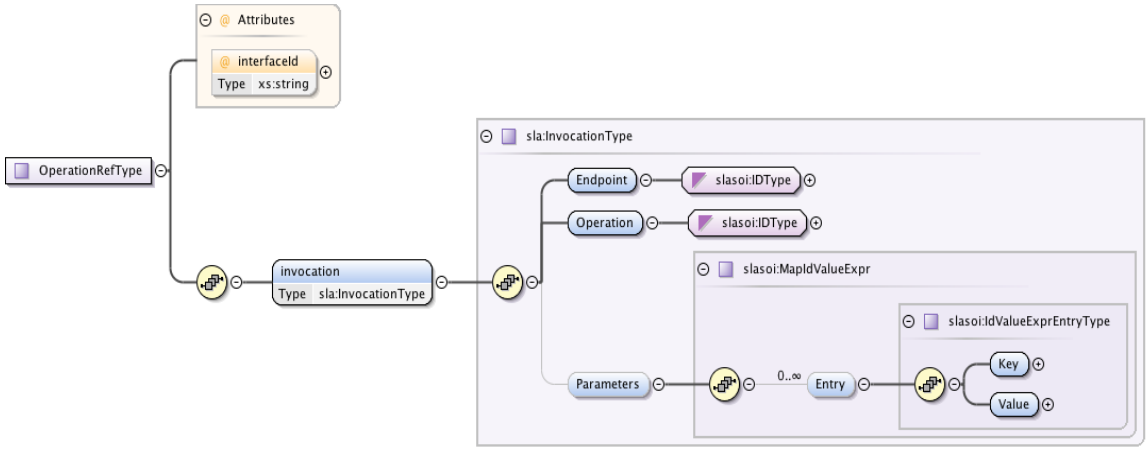


Figure 40 – Operation Ref Type

The XML schema representation of the *operationRefType* is shown below:

```
<xs:complexType name="OperationRefType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="invocation"
                type="operationCallType"/>
  </xs:sequence>
  <xs:attribute name="interfaceId" type="xs:string"/>
</xs:complexType>
```

According to *OperationRefType*, an operation invocation is described through:

- An attribute, called *interfaceId*, which refers to the interface that the operation to be invoked is a member of, and
- An *invocation* element. This element is specified by
  - a sub element, called *Endpoint*, which indicates the endpoint where the operation should be invoked,
  - a sub element, called *Operation*, which includes the id of the operation to be invoked, and

- an (optional list) of *Parameter* elements, which are used to provide the values for the different input parameters of the operation to be invoked. These parameters are indicated by key elements and their values as value elements.

Composite states are specified as instances of the type *CompositeState*, shown in Figure 41. As shown in the figure, a composite state element has three attributes, which are the same with the ones of the atomic transitions, and are the followings:

- A *stateId,* which uniquely identifies the state within a state transition model,
- A *name*, which provides the chosen name of the state, and
- A *description*, which can be used to provide a description of the intended meaning of the state.

In addition to these attributes, a *composite* state includes one or more *substate* elements.

A *substate* element is described as a state transition model itself, even if this sub-model has only one single *atomic* state itself. If a *composite* state has more than one *substate* elements, these elements are assumed to describe separate chunks of behaviour which are executed in parallel (i.e., AND- or parallel-decomposition). If there is only one *substate* element S, then the states of the state transition model that describes S are assumed to be disjunctive *substates*, (i.e., the entity can be at only one of these elements at any timepoint).

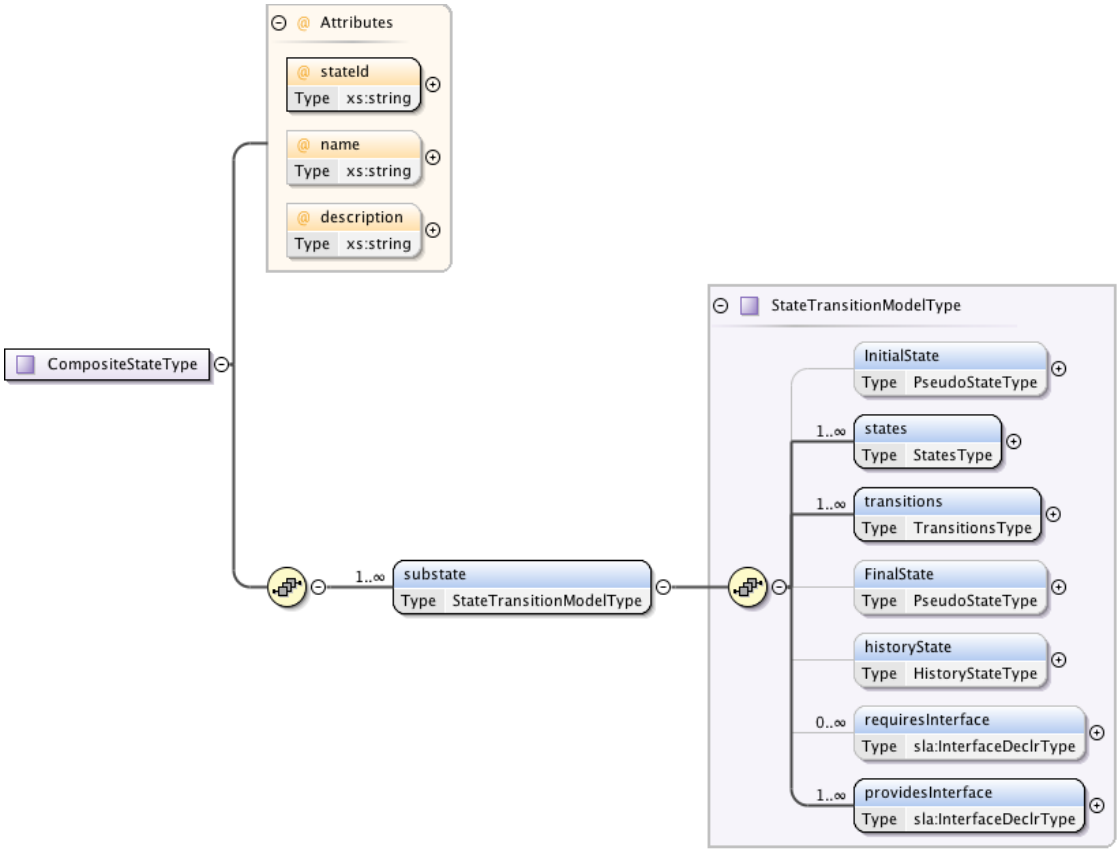


Figure 41 – Composite States Type

The definition of the type *CompositeStateType* in the certification model schema is provided below.

```
<xs:complexType name="CompositeStateType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="substate"
                type="StateTransitionModelType"/>
  </xs:sequence>
  <xs:attribute name="stateId" type="xs:string" use="required"/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="description" type="xs:string"/>
</xs:complexType>
```

4.3.1.10.2 TRANSITIONS SUB-ELEMENT

As shown in Figure 42, transitions in state transition models are described by a *Transitions* element, which is of a type *TransitionsType*. This element has one or more sub-elements, which are of type *IndividualTransitionType*.

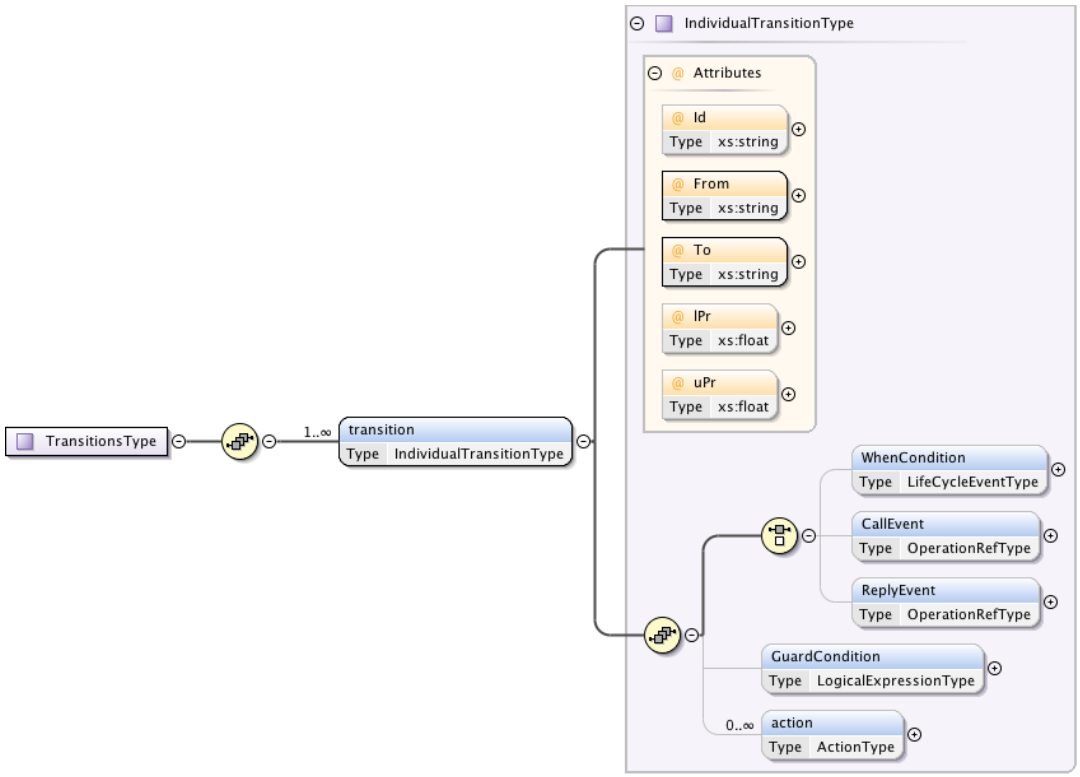


Figure 42 – Transition Element

The definition of type *IndividualTransitionType* in XML schema is shown below:

```
<xs:complexType name="IndividualTransitionType">
  <xs:sequence>
    <xsd:choice>
       <xs:element minOccurs="0" maxOccurs="1" name="WhenCondition"
                   type="LogicalExpressionType"/>
<!-- Trigerring conditions, i.e., conditions whose truth value change will
trigger the transition -->

<!-- Guard conditions, i.e., conditions which must be true for the
transition to be triggered -->
```

```
      <xs:element minOccurs="0" maxOccurs="1" name="CallEvent"
                  type="OperationRefType"/>
<!-- a call of an operation of the framework which should force the LC model
interpreter to take the transition, should be restricted to refer to the an
operation in one of the interfaces provided by the life cycle model -->


    </xsd:choice>
    <xs:element minOccurs="0" maxOccurs="1" name="GuardCondition"
                type="LogicalExpressionType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="action"
                type="ActionType"/>
  </xs:sequence>
  <xs:attribute name="From" type="xs:string" use="required"/>
  <xs:attribute name="To" type="xs:string" use="required"/>
  <xsd:attribute name="lPr" type="xs:float"/>
  <xsd:attribute name="uPr" type="xs:float"/>
</xs:complexType>
```

Each *IndividualTransitionType* element consists of:

- An attribute of type *String*, called *From*, which references the identifier of the state that the transition starts from,
- An attribute of type *String*, called *To*, which references the identifier of the state that the transition ends to, and
- Two more attributes to define the range of the expected relative frequency of undertaking a specific transition. These attributes are:
  - *lPr*, which is of a type *float* and is used to define the lower limit of the range, and
  - *uPr*, which is of a type *float* and is used to define the upper limit of the range.

  By using these two attributes the probability of a transition to be taken can be defined, which is usually used to express the *ExpectedSystemOperationModel* element of the

AssessmentScheme element. Thus, a range R(lPr, uPr) can be defined, which can be of the form **(**lPr, uPr**)**, **[**lPr, uPr**)**, **(**lPr, uPr**]** or **[**lpr, uPr**]** [2] .

Two types of events can trigger *transitions*:

- A system condition whose value is changed (such conditions are described by the sub-element *WhenCondition* in the above XML schema fragment), or
- A call event sent to the entity whose behaviour is described by the state transition model (such events are described by the sub-elements and *CallEvent* of a transition element).

WhenCondition elements are specified as elements of the type *LogicalExpressionType*, which is described in the next sub-section. A *call event* is specified as an element of *OperationRefType*.

Furthermore, transitions may have:

- A *GuardCondition* element, which is an element expressing a condition that must be true when an event that can trigger the transition occurs for the transition to be taken, and
- *Action* elements defining the actions that must be executed and completed before the transition is taken and the entity whose behaviour is described by the state transition model reaches the destination state of the transition.

4.3.1.10.3 LOGICALEXPRESSIONTYPE SUB-ELEMENT

A *logical expression* is defined as a condition, or logical combination of conditions, over the monitored evidence. Figure 43 shows the structure of the *LogicalExpressionType* and the *ConditionType*.

[2] The "[" and "]" denote a closed range at the lower and upper boundary respectively, and "(" and ")" denote an open range at the lower and upper boundary respectively.

```
<xs:complexType name="LogicalExpressionType">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Condition" type="ConditionType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
            name="WrappedExpression" type="WrappedExpressionType"/>
  </xs:sequence>
  <xs:attribute default="false" name="negated" type="xs:boolean"/>
</xs:complexType>

<xs:complexType name="ConditionType">
  <xs:sequence>
    <xs:element name="Operand1" type="RelationalOperandType"/>
    <xs:element name="Operand2" type="RelationalOperandType"/>
  </xs:sequence>
  <xs:attribute default="false" name="negated" type="xs:boolean"/>
  <xs:attribute name="relation" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="EQUAL-TO"/>
        <xs:enumeration value="NOT-EQUAL-TO"/>
        <xs:enumeration value="LESS-THAN"/>
        <xs:enumeration value="GREATER-THAN"/>
        <xs:enumeration value="LESS-THAN-EQUAL-TO"/>
        <xs:enumeration value="GREATER-THAN-EQUAL-TO"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

The XML schema for the specification of these two elements is shown below.

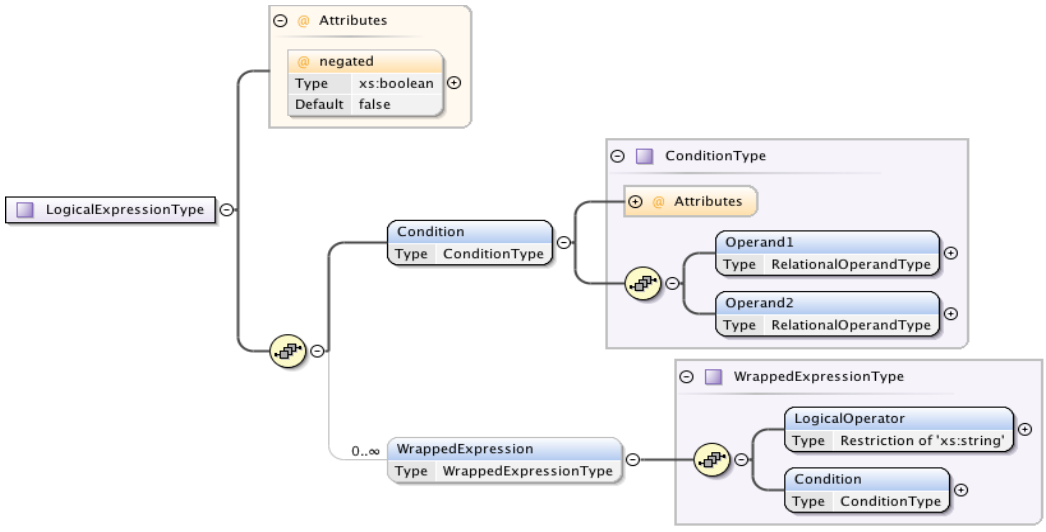


Figure 43 – Logical Expression Type

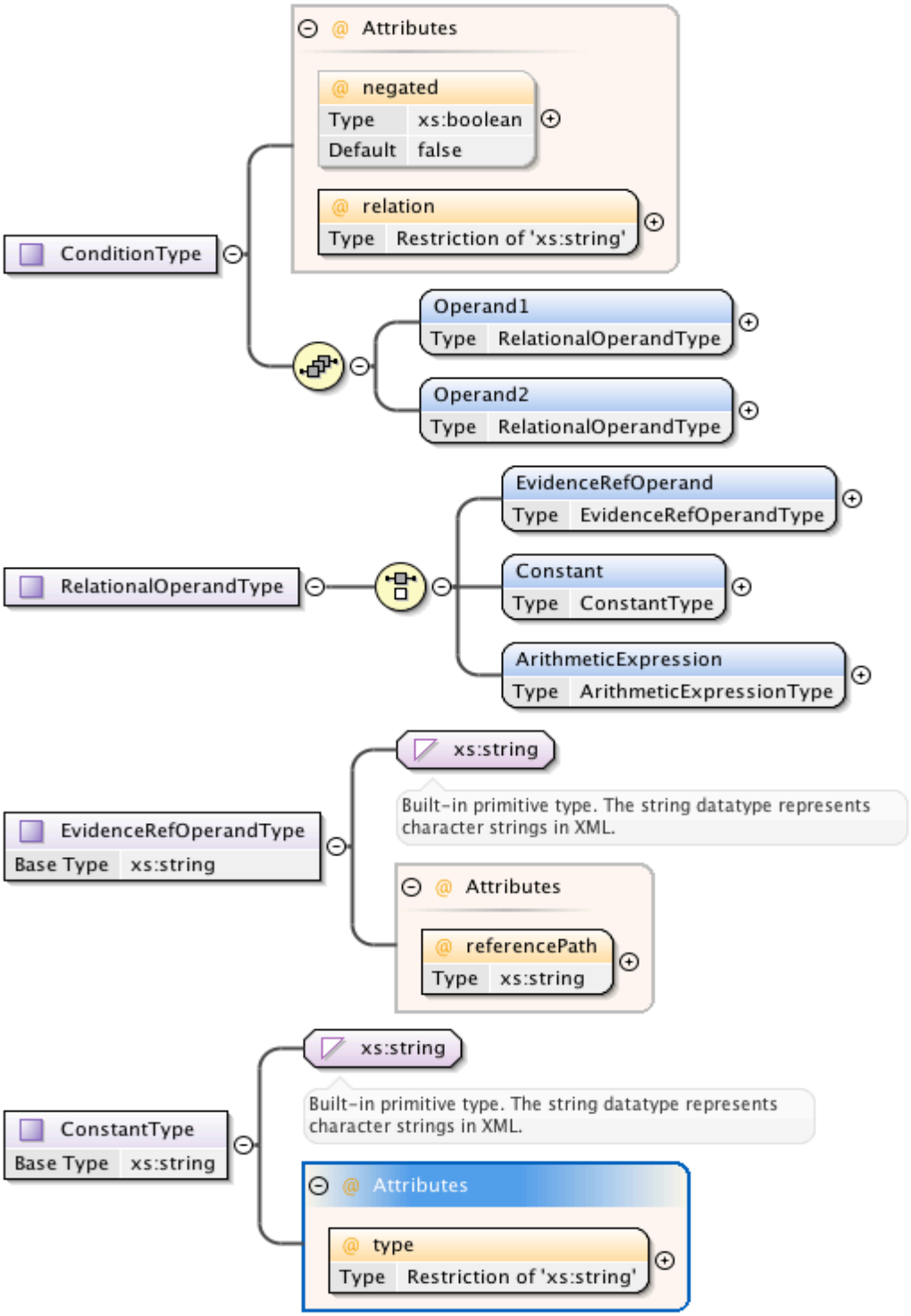


Figure 44 –Condition Type

The graphical representation of the *ConditionType* is presented in Figure 44 above. As shown in the figure a *Condition*:

- can be negated by defining the *Boolean* value of the *negated* attribute, and
- is defined as a *relational* operation, by specifying its *relation* attribute, which can take values such as *equalTo*, *notEqualTo*, *lessThan*, *greaterThan*, *lessThanEqualTo*, *greaterThanEqualTo* between two operands (the elements *operand1* and *operand2*). These operands can be evidence reference to:
  - *operand*,
  - *constants*, or
  - *arithmetic expressions*.

  An *evidence reference* operand, of a type *EvidenceRefOperandType*, can be used to refer to monitorable evidence specified in the certification model.

  Constant operand, of a type *ConstantType*, can be used to specify a numerical or string constant value.

The XML schema for the specification of *Condition* elements is shown below.

```
<xs:complexType name="ConditionType">
  <xs:sequence>
    <xs:element name="Operand1" type="RelationalOperandType"/>
    <xs:element name="Operand2" type="RelationalOperandType"/>
  </xs:sequence>
  <xs:attribute default="false" name="negated" type="xs:boolean"/>
  <xs:attribute name="relation" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="EQUAL-TO"/>
        <xs:enumeration value="NOT-EQUAL-TO"/>
        <xs:enumeration value="LESS-THAN"/>
        <xs:enumeration value="GREATER-THAN"/>
        <xs:enumeration value="LESS-THAN-EQUAL-TO"/>
        <xs:enumeration value="GREATER-THAN-EQUAL-TO"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
```

```
</xs:complexType>

<xs:complexType name="RelationalOperandType">
  <xs:choice>
    <xs:element name="EvidenceRefOperand" type="EvidenceRefOperandType"/>
    <xs:element name="Constant" type="ConstantType"/>
    <xs:element name="ArithmeticExpression" type="ArithmeticExpressionType"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="EvidenceRefOperandType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="referencePath" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="ConstantType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="NUMERICAL"/>
            <xs:enumeration value="STRING"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

An example of *Condition* is shown below. This condition specifies that the value of the variable *nofcalls* should be equal than zero.

```
<Condition>
  <evidenceCondition relation="EQUAL-TO">
    <Operand1>
      <EvidenceRefOperand
       referencePath="//VariableDeclr/Var/[text()='nofcalls']"/>
    </Operand1>
    <Operand2>
      <Constant type="NUMERICAL">0</Constant>
    </Operand2>
  </evidenceCondition>
</Condition>
```

Arithmetic expression operand defines computations over the values of monitorable evidence in the certification model. The structure of the *arithmetic expression* is shown in Figure 45.

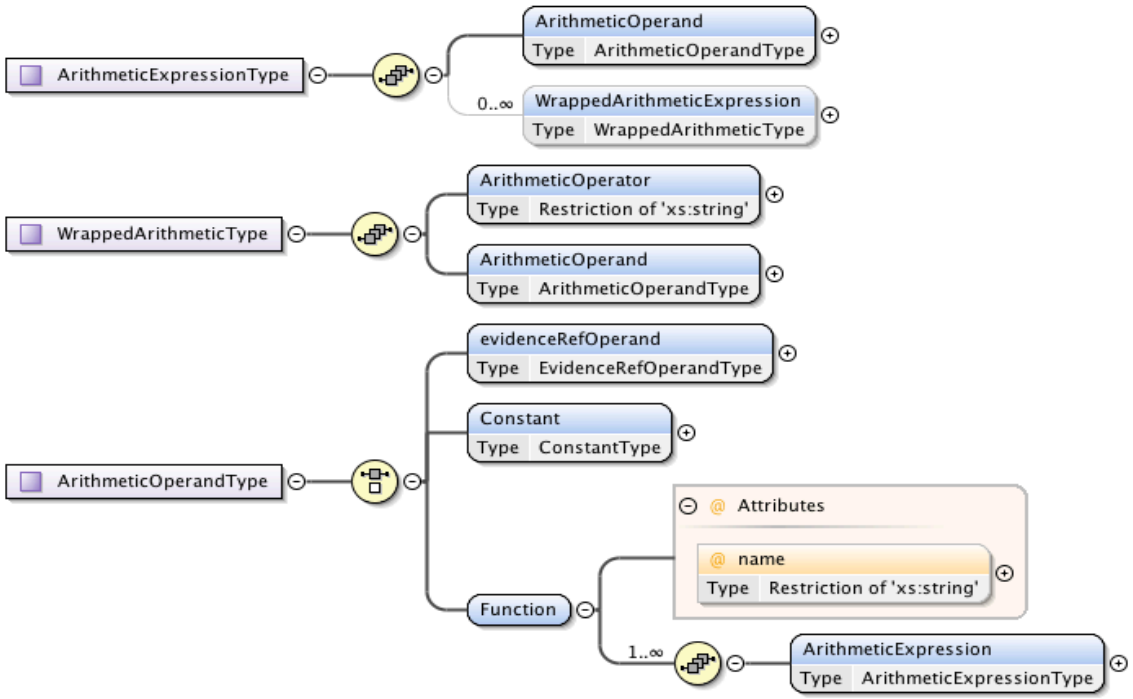


Figure 45 – Arithmetic Expression Type

As shown in the figure, an *arithmetic expression* is defined as a sequence of *arithmetic operands* or other nested arithmetic expressions of the type *WrappedArithmeticExpressions*, which are connected by *arithmetic operators*. The *arithmetic operators* are:

- addition (*plus*),
- subtraction (*minus*),
- multiplication (*multiply*), and
- division (*divide*) operators.

As shown in XML schema, the operands can be:

- *evidence reference* operands,

- *constants*, or
- *functions*.

A *function* supports the execution of a complex computation over a series of arguments. The results of these computations are numerical values that can be used as an operand in an arithmetic expression. A *function* has a name and a sequence of one or more arguments. Each of these arguments may be an *arithmetic expression*. The currently supported functions are *MIN* and *MAX*, which choose the minimum or maximum value of these expressions supplied.

The XML schema for the arithmetic expression element is shown below.

```
<xs:complexType name="ArithmeticExpressionType">
  <xs:sequence>
    <xs:element name="ArithmeticOperand" type="ArithmeticOperandType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
            name="WrappedArithmeticExpression" type="WrappedArithmeticType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="WrappedArithmeticType">
  <xs:sequence>
    <xs:element name="ArithmeticOperator">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="PLUS"/>
          <xs:enumeration value="MINUS"/>
          <xs:enumeration value="MULTIPLY"/>
          <xs:enumeration value="DIVIDE"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ArithmeticOperand" type="ArithmeticOperandType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ArithmeticOperandType">
  <xs:choice>
    <xs:element name="evidenceRefOperand" type="EvidenceRefOperandType"/>
    <xs:element name="Constant" type="ConstantType"/>
    <xs:element name="Function">
      <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
          <xs:element name="ArithmeticExpression"
                      type="ArithmeticExpressionType"/>
        </xs:sequence>
```

```
        <xs:attribute name="name">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="MAX"/>
              <xs:enumeration value="MIN"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

An example of logical expression is shown below. This example specifies a logical expression that checks that the event sufficiency condition (see the first condition in the logical expression) is satisfied, that no conflict has been detected (see the second condition inside the first nested logical expression), and that the value of the variable nofcalls is equal to zero (see the condition in the second nested logical expression).

```
<LogicalExpression negated="false">
  <Condition>
    <evidenceSufficiencyCondition>1011</evidenceSufficiencyCondition>
  </Condition>
  <LogicalOperator>AND</LogicalOperator>
  <LogicalExpression negated="true"/>
    <Condition>
      <conflictCondition>1100</conflictCondition>
    </Condition>
  </LogicalExpression>
  <LogicalOperator>AND</LogicalOperator>
  <LogicalExpression negated="true"/>
    <Condition>
      <evidenceCondition relation="EQUAL-TO">
        <Operand1>
          <EvidenceRefOperand
            referencePath="//VariableDeclr/Var/[text()='nofcalls']"/>
        </Operand1>
        <Operand2>
          <Constant type="NUMERICAL">0</Constant>
        </Operand2>
      </evidenceCondition>
    </Condition>
  </LogicalExpression>
</LogicalExpression>
```

Chapter Five

CASE STUDIES AND EXAMPLE CERTIFICATION MODELS

5.1 OVERVIEW

As explained in the previous chapter, a certifier should define a certification model in order to carry out the assessment and potentially certify a security property for a cloud service. In Chapter 4, we introduced the language (XML schema) for specifying certification models and in this chapter we give three examples of such models.

The first example is a CM that can be used to assess and certify whether a cloud service satisfies the security property of non-repudiation. The next two certification models are defined for systems offering cloud services that have been used as drivers for developing our approach, namely a health care system and a cloud enabled smart city application.

It should be noted that the CM examples are defined using the XML schema introduced in the previous chapter with the exception of the model assertions. The latter are provided in the abstract syntax of EC-Assertion, i.e., the language of the EVEREST monitoring tool, for simplicity as the full listing of assertions in XML would be long. The full listing of all assertions, however, is provided in Appendices A, B and C.

5.2 NON-REPUDIATION PROTOCOL FOR CLOUD SERVICES

In our example non-repudiation is a property that refers to non-repudiation of data transfer. More specifically, it requires that when a data owner or a consumer sends a request to a cloud provider, to either upload or download data respectively, these transactions should be conducted in a way such that none of the involved parties could deny having participated in a part or the

whole of this communication. Several protocols have been proposed so far, in order to realise the non-repudiation security property, such as [93][111][159][165][236][237]. The basic principle that underpins these protocols is that along with a data-uploading (downloading) request, the data owner (consumer) sends a "Non-Repudiation of Origin" (*NRO*) token, which is the proof of sending the request. In return, they will expect to receive evidence of "Non-Repudiation of Receipt" (*NRR*) from the cloud provider, acknowledging that the specific request was received.

The example certification model that we introduce in this chapter aims to certify the adherence to a Non Repudiation (NR) Protocol for cloud services, presented in [93] and, therefore, the preservation of the NR security property by the service. To enable the reader understand this certification model, we first give an overview of the underpinning NR protocol.

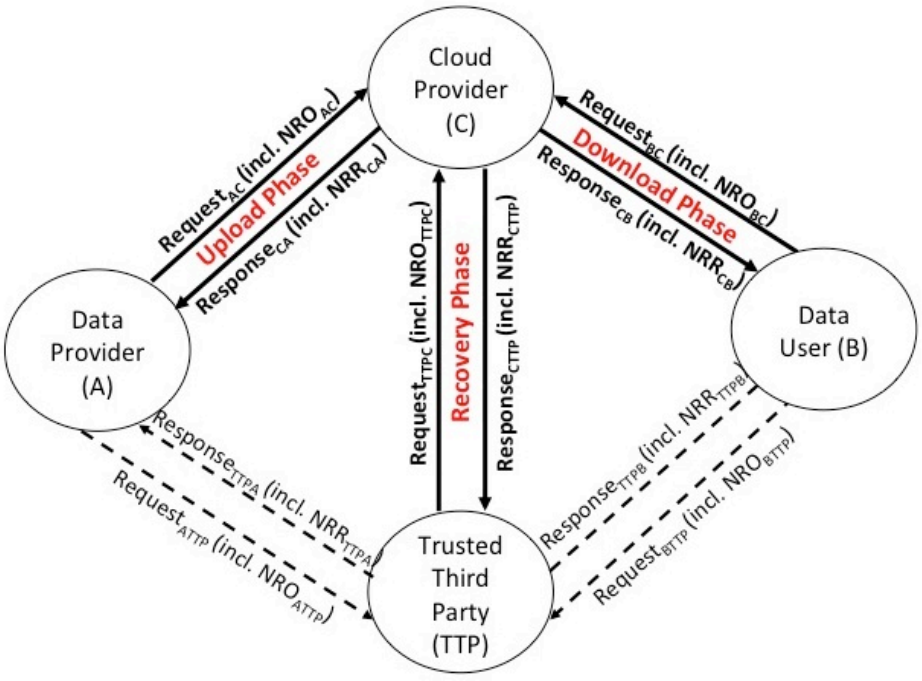


Figure 46 – Non-Repudiation Protocol for Cloud Services

This protocol involves four different parties, which are:

- A data owner/provider (*A*),

- A data user (*B*),
- The Cloud Provider (*C*), and
- A Trusted Third Party (*TTP*).

These parties interact through three different phases of the protocol, which are shown in Figure 46. These phases are:

- The data upload phase,
- The data download phase, and
- A recovery phase.

Before describing these phases, we provide some basic definitions necessary for understanding their meaning. Therefore, we have:

NRO: Evidence of Non-Repudiation of Origin, sent by a sender to a receiver. The receiver will hold this evidence as a proof if the sender denies having sent the message.

NRR: Evidence of Non-Repudiation of Receipt, sent by the receiver to the sender. The sender will hold this evidence as a proof if the receiver denies having received the message.

f_M: Flag indicating the intended purpose of a message *M*.

l: Unique label chosen by *A* to link all messages.

M: Message sent from a sender to a receiver.

H(M): Hash function applied to message *M*.

K: Message key defined by the sender.

B_L: Group of data users *Bi* who are authorised to download message *M* and are capable of decrypting it.

Seqi: Unique sequence number of each message.

$EG_B()$: Group encryption scheme known only to *BL* group.

$E_X(Y)$: Asymmetric encryption of message *Y* produced by party *X*'s public key.

$S_X(Y)$: signature of message *Y* produced by *X*'s private key.

The three different phases of the NR protocol are described below.

1 Upload Phase

In this phase the data owner *A* sends a request to the cloud provider *C*, for uploading data. Firstly, *A* encrypts a message *M* (i.e., the data) with a key *K* and generates two different NROs, the NRO_{AB} and the NRO_{AC}. The NRO_{AB} will be used by data users *B* to get the key *K* and the $S_A(H(M))$. The *K* is required to decrypt the *M* and the $S_A(H(M))$ to verify the data integrity after downloading *M* from *C*. *A* encrypts the NRO_{AB} by using the group encryption scheme $EG_B()$ to guarantee that only the intended recipients of the B_L can have access and decipher NRO_{AB} and *M*. NRO_{AC} is the proof of evidence that *A* sent the request to *C* and is encrypted with *C*'s public key. This step is defined as:

$$A \rightarrow C: \quad RQS_{AC} = \{f_{RQS_{AC}}, l, A, C, TTP, H(M), H(B_L), Seq_1, T_{g1}, T_1, EG_B(NRO_{AB}), E_C(NRO_{AC})\}$$

Where:

- T_1 is the maximum time that the sender will wait for an NRR to RQS_{AC}.
- T_{g1} is the time of the generation of RQS_{AC}.
- NRO_{AB} is an NRO sent from A to B users through C. It is visible to all B_L recipients, but not to C itself.
- $NRO_{AB} = \{K, l, S_A(H(M))\}$
- NRO_{AC} is an NRO sent from A to C, defined as
- $NRO_{AC}=\{S_A(H(M), H(B_L), EG_B(NRO_{AB}), H(l, Seq_1, Tg_1, T_1))\}$.

When *C* receives a RQS_{AC} it must produce a response to *A*. This step is defined as:

$C \rightarrow A$: $RSP_{CA} = \{f_{RSP_{CA}}, l, A, C, TTP, H(M), H(B_L), Seq_2, T_{g2}, T_S, E_A(NRR_{CA})\}$

Where:

- T_S is time when data is stored
- T_{g2} is the time of the generation of RSP_{AC}.
- NRR_{CA} is the NRR sent from C to A, defined as
- $NRR_{CA} = \{S_C(H(M)), S_C(H(l, Seq_2, Tg_2, T_S, NRO_{AC}))\}$.

2 Download Phase

In this phase, the data user *B* downloads data from the cloud provider *C*. To do so, *B* sends a request with an NRO_{BC} to *C*. The request should include *B*'s identity to enable *C* verify whether the *B* is authorised to download the requested data. This is done by checking *B*'s identity against the B_L, which was received for the *M* that was sent from the data owner *A*. If *B* is in B_L, *C* will send the requested data with the encrypted NRO_{AB} ($EG_B(NRO_{AB})$) received from *A*, along with its own non-repudiation evidence NRR_{CB}. These exchanges are defined below:

$B_i \rightarrow C$: $RQS_{BiC} = \{f_{RQS_{BiC}}, l_i, A, C, B_i, TTP, Seq_3, T_{g3}, T_2, E_C(NRO_{BC})\}$

Where:

- $l_i = H(A, C, B_i, TTP)$
- NRO_{BC} is the NRO sent from B to C, defined as
- $NRO_{BC} = \{S_B(H(l_i, A, C, TTP, Seq_3, T_{g3}, T_2))\}$.

The response produced by *C* whilst receiving a download request from *B* is:

$C \rightarrow B_i$: $RSP_{CBi} = \{f_{RSP_{CBi}}, l, A, C, B_i, TTP, H(M), Seq_4, T_{g4}, EG_B(NRO_{AB}), E_{B_i}(NRR_{CB})\}$

Where:

- NRR_{CB} is the NRR sent from C to B, defined as

$NRR_{CB}=\{S_C(H(M)), S_C(EG_B(NRO_{AB})), S_C(H(l, Seq_4, Tg_4))\}$.

When *B* gets the data and the EG_B from *C*, it will obtain the *K* and *H(Data)* by decrypting the NRO_{AB} and will check the integrity of the data and the validity of NRR_{CB}.

3 Resolution of Disputation

This phase refers to the cases where there are some disputations in the communication between users and the cloud provider. For instance, if *A* does not receive the expected response from the *C*, it sends a request to *TTP* with its identification and the NRO_{AC}. Subsequently, the *TTP* will send a request to *C* that will include the NRO_{AC} sent from *A* and its own NRO_{TC} and *C* should respond with an NRR_{TC} that will include the corresponding NRR_{CA}. These exchanges are defined bellow:

$TTP \rightarrow C: RQS_{TC} = \{f_{RQS_{TC}}, l, A, C, TTP, Seq_5, T_{g5}, T_3, E_C(NRO_{AC}), E_C(NRO_{TC})\}$

$C \rightarrow TTP: RSP_{CT} = \{f_{RSP_{CT}}, l, A, C, TTP, Seq_6, T_{g6}, T_S, E_A(NRR_{CA}), E_T(NRR_{CT})\}$

Where:

- T_3 is the maximum time that the sender will wait for an NRR to RQS_{TC}.
- Tg_5 (Tg_6) is the time of the generation of RQS_{TC} (RSP_{CT}).
- T_S is the time when data was stored by C.
- NRO_{TC} is the NRO sent from the TTP to C to resolve a disputation regarding an uploading session of A, defined as $NRO_{TC}=\{S_T(H(l, A, C, TTP, Seq_5, Tg_5, T_5, E_C(NRO_{AC})))\}$.
- NRR_{CT} is sent from C to TTP, defined as

  $NRR_{CT}=\{S_C(H(l, Seq_5, Tg_5, NRR_{CA}))\}$.

5.2.1 CERTIFICATION MODEL SPECIFICATION

In the following, we provide definitions of all the individual elements of the NR certification model. For simplicity reasons we focus on only one of the three phases, i.e., the upload phase. It should be noted, however, that the specification of CM model elements for the other two phases of the NR protocol is analogous.

5.2.1.1 MODEL_ID ELEMENT

The first element of the CM is the unique identifier of the specific CM. In the current example, the CM has the value "*cm:id:monitoring:0001*". The format selected is to define except of the Id, also its type that is "monitoring" in our example.

```
<Model_Id>cm:id:monitoring:0001</Model_Id>
```

5.2.1.2 SIGNATURE ELEMENT

As explained in the previous chapter, each CM should provide the information regarding the certification authority (CA) that defines and then signs the generated certificates. In the current example the signature of the certifier is defined as "City" and its role is "CA", as shown below.

```
<Signature>
   <Name>City</name>
   <Role>CA</Role>
</Signature>
```

5.2.1.3 TOC ELEMENT

The *ToC* element defines the target of certification of the certification model. As explained in the previous chapter, it consists of an attribute and two sub-elements that refer to the interfaces that are either provided or required.

The example below illustrates how this element should be defined. It shows that the TOC has an ID "*id1001*" provides an interface with the name "*cloudinterface*" that has three operations,

namely *rqsac, rqsbc* and *rqstc*, which are the requests that can be made in each of the three different phases of the NR protocol. Each of these operations gets as inputs *data* of a type *url* and the *rsqac* has also a second input *t* of the type url. These inputs are external variables (*forMatching* = *true*) and their value will not be the same throughout all instances (*persistent* = *false*). The XML example for the ToC is provided below

```
<TOC id="id1001">
  <providesInterface>
    <ID>001</ID>
    <ProviderRef>provider001</ProviderRef>
    <Interface>
      <InterfaceSpec>
        <Name>cloudinterface</Name>
        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0001</OperationId>
          <operationName>rqsac</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
          <inputVariable forMatching="false" persistent="true">
            <varName>t</varName>
            <varType>url</varType>
          </inputVariable>
        </Operation>

        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0002</OperationId>
          <operationName>rqsbc</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
        </Operation>

        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0003</OperationId>
          <operationName>rqstc</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
        </Operation>
      </InterfaceSpec>
```

```
    </Interface>
  </providesInterface>
</TOC>
```

5.2.1.4 SECURITYPROPERTY ELEMENT

The definition of the Non-repudiation (NR) security property in the NR certification model is given by the following XML security property element:

```
<SecurityProperty SecurityPropertyId="0001"
 SecurityPropertyDefinition="AIS:non-repudiation:non-repudiation-of-origin"
Vocabulary="CSA"
        ShortName="AIS:non-repudiation:non-repudiation-of-origin">
  <sProperty class="http://www.cumulus-project.eu">
    <propertyPerformance>
      <propertyPerformanceRow>
        <propertyPerformanceCell name="verified">true
        </propertyPerformanceCell>
      </propertyPerformanceRow>
    </propertyPerformance>
    <propertyParameterList/>
  </sProperty>

  <Assertion ID="AS001">
    <InterfaceDeclr> … </InterfaceDeclr>
    <VariableDeclr> … </VariableDeclr>
    <Guaranteed> … </Guaranteed>
  </Assertion>
</SecurityProperty>
```

As shown in the above listing,, the attributes of the security property that defines NR are:

- The "Security*PropertyId = 0001*",
- The definition of the property, or in other words its name "*SecurityPropertyDefinition = AIS:non-repudiation:non-repudiation-of-origin"*,
- The vocabulary used to define the property, which in our case we used the vocabulary defined by the Cloud Security Alliance "*Vocabulary=CSA*", and
- A short version of its name "*ShortName=AIS:non-repudiation:non-repudiation-of-origin"*.

The following sub-elements concerning the assertion of the security property, which has an Id = *AS001*, are defined in the following section.

5.2.1.4.1 INTERFACEDECLR SUB-ELEMENT

In this sub-element the interfaces offered by the service that are related to the security property are defined. In the current example for the NR property the interface provided is the *provider001* provider. Moreover, the operations and their inputs that should be checked are also defined, which are the *rqsac*, *rqsbc, rqstc* and *rspct*. All inputs are external and are not static according to the Boolean values of the attributes *forMatching* and *persistent.*

Below the definition of the *InterfaceDeclr* sub-element for the NR property is provided.

```
<InterfaceDeclr>
  <ID>001</ID>
  <ProviderRef>provider001</ProviderRef>
  <Interface>
    <InterfaceSpec>
      <Name>cloudinterface</Name>
      <Operation>
        <interfaceId>c</interfaceId>
        <OperationId>id0004</OperationId>
        <operationName>rqsac</operationName>
        <inputVariable forMatching="false" persistent="true">
          <varName>data</varName>
          <varType>url</varType>
        </inputVariable>
        <inputVariable>
          <varName>seq</varName>
          <varType>string</varType>
        </inputVariable>
        <inputVariable forMatching="false" persistent="true">
          <varName>t</varName>
          <varType>url</varType>
        </inputVariable>
      </Operation>
      <Operation>
        <interfaceId>c</interfaceId>
        <OperationId>id0005</OperationId>
        <operationName>rqsbc</operationName>
        <inputVariable forMatching="false" persistent="true">
          <varName>data</varName>
```

```
          <varType>url</varType>
        </inputVariable>
        <inputVariable>
          <varName>seq</varName>
          <varType>string</varType>
        </inputVariable>
      </Operation>
      <Operation>
        <interfaceId>c</interfaceId>
        <OperationId>id0007</OperationId>
        <operationName>rqstc</operationName>
        <inputVariable forMatching="false" persistent="true">
          <varName>data</varName>
          <varType>url</varType>
        </inputVariable>
        <inputVariable>
          <varName>seq</varName>
          <varType>string</varType>
        </inputVariable>
      </Operation>
    </InterfaceSpec>
  </Interface>
</InterfaceDeclr>
```

5.2.1.4.2 GUARANTEED SUB-ELEMENT

The monitorable specification of the NR property is provided by the *Guaranteed* sub-element of the security property. This element is specified in EC-Assertion as[3]:

```
Rule R1:

Happens(e(_id1,_sender1,_receiver1, RQS_AC(_seqrq),_receiver1), _tAReq,
[_t0, _t0])
∧ ¬HoldsAt(ResUpReq(_seqrq, _seqrs,_t), _t0) ⇒
Happens(e(_id2,_receiver1,_sender1,RSP_CA(_seqrs),_receiver1), _tg2,
[_t0,_t0+100])
```

This rule states that when there is a call event *RQSac,* with a variable _seqrq that uniquely specifies the sequence of the request event, at the specific time instance *t0*, which is a request sent from the data owner *A* for uploading data to a cloud provider *C*, and as long as there is no

[3] This assertion is specified in XML in Appendix A

previous request received by the cloud provider *C* from *A* with the same sequence number that has not been responded (*ResUpReq* fluent not holds) until *t0* (precondition), a reply event *RSPca* should occur from *C* to *A* at the time tg2, which should be in the time range between *t0* and *t0* plus 100 msec.

Since the assertion in the rule needs to be checked against information that will arise after the occurrence of the event that will trigger the check, the *Guaranteed* has the type "*Future_Formula"*. In order to determine the conditions under which the assertion should be checked, the *precondition* element is defined, in which it is stated that a request event should happen and a fluent should not be hold at the time of the occurrence of the event. Thus, an event of the type *call* is defined in the precondition, and its attribute *unconstrained* is true, as it is an instantaneous event. Furthermore, a *WrappedCondition* element is used in the precondition with the logical *Operator* AND, because in the rule there is a combined condition regarding the occurrence of an event and the state of the fluent *ResUpReq*. In order to define that the fluent does not hold at the time t1 when the event occurred, we use the *holdsAt* element with the attribute *negetated = "true"*.

In the *postcondition* element, the reply event that should occur at the time t2 from the Cloud Provider is defined. This event is of a type *reply*, the event condition is *unconstrained = false*, as this event has a predefined time Range, and for expressing the time range [t1, t1+100] the *plus* operator is used in the *toTime* element. Due to the fact that the reply event has a predefined time range that is expected to occur, t2 is defined as *existential* time variable type.

Below the XML definition of the above rule is given below.

```
<Guaranteed ID="gt1" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <quantification>
    <quantifier>existential</quantifier>
    <timeVariable>
      <varName>t2</varName>
      <varType>TimeVariable</varType>
```

```
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="gt1-ac1">
      <eventCondition unconstrained="true">
        <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>CP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>upload</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
              <value>REQ-B</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>seqrq</varName>
              <varType>int</varType>
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
```

```
          </fromTime>
          <toTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
    <WrappedCondition>
      <operator>and</operator>
      <assertionCondition>
        <atomicCondition conditionID="gt1-ac2">
          <stateCondition>
            <holdsAt negated="false">
              <state name="ResUpReq">
                <argument>
                  <variable persistent="false" forMatching="true">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </variable>
                </argument>
                <argument>
                  <variable forMatching="true" persistent="false">
                    <varName>seqrs</varName>
                    <varType>int</varType>
                  </variable>
                </argument>
              </state>
              <timeVar>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </holdsAt>
          </stateCondition>
        </atomicCondition>
      </assertionCondition>
    </WrappedCondition>
  </precondition>
  <postcondition>
    <atomicCondition conditionID="gt1-ac3">
      <eventCondition unconstrained="true">
        <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID1</varName>
          </eventID>
          <reply>
            <interfaceId>CP</interfaceId>
            <OperationId>2</OperationId>
```

```
      <operationName>upload</operationName>
      <outputVariable forMatching="true" persistent="false">
        <varName>status2</varName>
        <varType>OpStatus</varType>
        <value>RES-B</value>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>receiver1</varName>
        <varType>Entity</varType>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>sender1</varName>
        <varType>Entity</varType>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>source1</varName>
        <varType>Entity</varType>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>serviceId</varName>
        <varType>string</varType>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>seqrs</varName>
        <varType>int</varType>
      </outputVariable>
      <outputVariable forMatching="true" persistent="false">
        <varName>seqrq</varName>
        <varType>int</varType>
      </outputVariable>
    </reply>
    <tVar>
      <timeVar>
        <varName>t2</varName>
        <varType>TimeVariable</varType>
      </timeVar>
    </tVar>
    <fromTime>
      <time>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
      </time>
    </fromTime>
    <toTime>
      <time>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
      </time>
      <Expression>
        <plus>100</plus>
```

```
            </Expression>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
  </postcondition>
</Guaranteed>
```

To initiate or update the fluent, the following assumption is made. This assumption states that when an upload request is made at t1 and if the fluent *ResUpReq* does not hold at the time t1 and if a reply to the request is made at t2, then the reply event will initialise the fluent at time t2, which will keep the information about the sequence number of the request event and its reply event.

Assumption 1:
R1.A2: Happens(e(_id1,_sender1,_receiver1,RQS_{AC}(_seqrq),_receiver1),_t1, [_t1,_t1])
∧ ¬ HoldsAt(ResUpReq(_seqrq,_seqrs, _t), _t1)
∧ Happens(e(_id2, _receiver1, _sender1, RSP_{CA}(_seqrs), _receiver1),_t2, [_t1, _t1+100) ⇒
Initiates(e(_id2, _receiver1, _sender1, RSP_{CA}(_seqrs), _receiver1), ResUpReq(_seqrq, _seqrs, _t2), _t2)

The element of the above assumption is expressed in the same way as the rule assertion. XML definition can be found in Appendix A, with the *Guaranteed Id= "gt2"*.

5.2.1.5 ASSESSMENTSCHEME ELEMENT

According to the certification model, in the assessment scheme element there are four sub-elements that can be defined. These are:

- Evidence Sufficiency Condition element,
- Expiration Condition,
- Conflict, and
- Anomalies.

5.2.1.5.1 EVIDENCE SUFFICIENCY CONDITION ELEMENT

The *EvidenceSufficiencyCondition* element defines all the conditions regarding the sufficient evidence that must be collected, in order to be able to issue a certificate. Below we provide examples of how to define this element, as there are different ways to express it.

The first example corresponds to a monitoring period condition and states that the minimum period that a cloud service should be monitored before issuing a certificate is 3 days.

```
<EvidenceSufficiencyCondition Id="1011">
     <MonitoringPeriodCondition minMonitoredPeriod="3" periodUnit="days"/>
</EvidenceSufficiencyCondition>
```

The second example corresponds to a monitoring events condition and states that a minimum of at least 5000 events should be monitored before issuing a certificate.

```
<EvidenceSufficiencyCondition Id="1012">
            <MonitoringEventsCondition eventsNo="5000"/>
</EvidenceSufficiencyCondition>
```

The third example corresponds to a sufficiency condition defined through an expected system behaviour model. Our example refers to the monitoring based certification model for the NR property and is a model of the expected behaviour of the ToC for this property, i.e., the cloud storage service C that should be certified that realises the NR protocol and, through it, satisfies the NR security property.

A graphical representation of this model is shown in Figure 47. As the model shows the cloud storage service C initially gets to state1 and at this state it may respond to:

- Calls of the operation *RQSac(nroac)* from a data owner a to upload data along with an NRO for it. The model specifies that the frequency of C receiving such a call whilst being in state 1 is Pr[01, 0.5), meaning that at least 10% and below 50% of the events should be of this type of calls, which make C move to state 2. Whilst at state 2 there should be a response event *RSPca(nrrca)* with a frequency range Pr[1,1] which states that there

should be a response event for the call event that lead C to state 2, in order for C to move back to state 1.

- Calls of the operation *RQSbc(nrobc)* from a data user B a to upload data along with an NRO for it. The model specifies that the probability of C receiving such a call whilst being in state 1 is Pr [0.5, 1), which denotes that at least 50% of the events should be of this type of calls. This event will make C move to state 4. Whilst at state 4 there should be a response event *RSPcb(nrrcb)* with a frequency range Pr[1,1] which states that there should be a response event for the call event that lead C to state 4, in order for C to move back to state 1.
- Calls of the operation *RQStc(nrotc)* from a TTP for a resolution of a disputation. The model specifies that the probability of C receiving such a call whilst being in state 1 is Pr (0.0, 0.2), which states that up to below 20% of the events should be of this type. This event will make C move to state 3. Whilst at state 3 there should be a response event *RSPct(nrrct)* with a frequency range Pr[1,1] which states that there should be a response event for the call event that lead C to state 3, in order for C to move back to state 1.

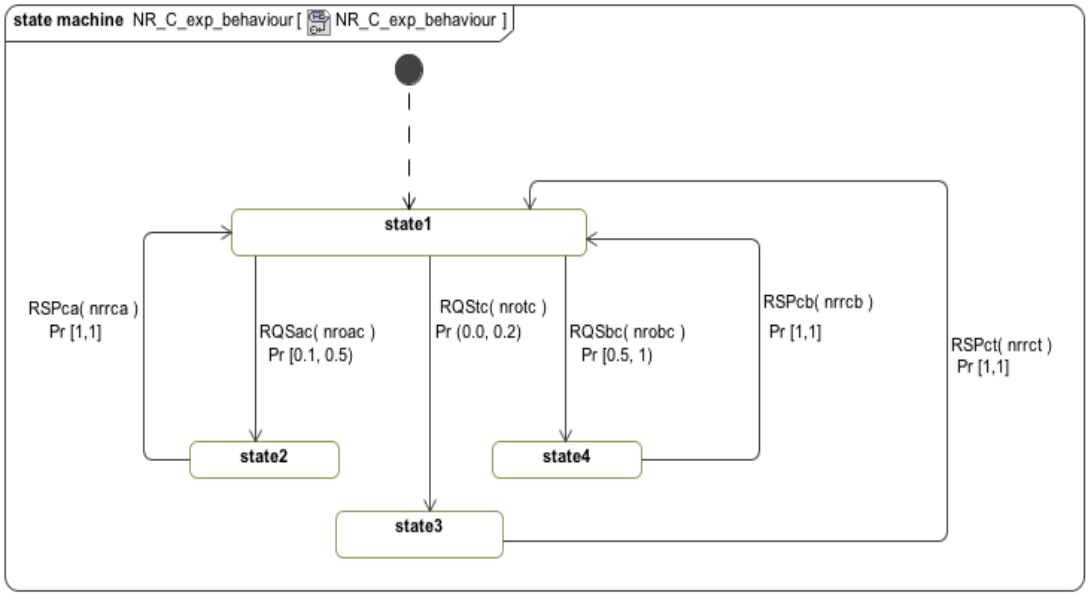


Figure 47 – Example of expected TOC Behaviour Model

Below an XML example of the *ExpectedSystemOperationModel* is given, to show the way such state transition model is defined.

```
<ExpectedSystemOperationModel>
  <InitialState stateId="s1" name="state1"/>
  <states>
    <state>
      <atomicState stateId="s2" name="state2"/>
    </state>
    <state>
      <atomicState stateId="s3" name="state3"/>
    </state>
    <state>
      <atomicState stateId="s4" name="state4"/>
    </state>
  </states>
  <transitions>
    <transition Id="t1" From="s1" To="s2" lPr="0.1" uPr="0.5">
      <CallEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>upload</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>DP</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
            </inputVariable>
          </execute>
        </invocation>
      </CallEvent>
    </transition>
    <transition Id="t2" From="s2" To="s1" lPr="1" uPr="1">
      <ReplyEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>upload</operationName>
            <outputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
```

```
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>DP</value>
            </outputVariable>
          </execute>
        </invocation>
      </ReplyEvent>
    </transition>
    <transition Id="t3" From="s1" To="s3" lPR="0.0" uPr="0.2">
      <CallEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>2</OperationId>
            <operationName>download</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>DC</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
            </inputVariable>
          </execute>
        </invocation>
      </CallEvent>
    </transition>
    <transition Id="t4" From="s3" To="s1" lPr="1" uPr="1">
      <ReplyEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>2</OperationId>
            <operationName>download</operationName>
            <outputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>DC</value>
            </outputVariable>
          </execute>
        </invocation>
```

```
      </ReplyEvent>
    </transition>
    <transition Id="t5" From="s1" To="s4" lPr="0.5" uPr="1">
      <CallEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>3</OperationId>
            <operationName>resolution</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>TTP</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
            </inputVariable>
          </execute>
        </invocation>
      </CallEvent>
    </transition>
    <transition Id="t6" From="s4" To="s1" lPr="1" uPr="1">
      <ReplyEvent>
        <invocation>
          <execute>
            <interfaceId>CP</interfaceId>
            <OperationId>3</OperationId>
            <operationName>resolution</operationName>
            <outputVariable forMatching="true" persistent="false">
              <varName>receiver</varName>
              <varType>String</varType>
              <value>CP</value>
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>sender</varName>
              <varType>String</varType>
              <value>TTP</value>
            </outputVariable>
          </execute>
        </invocation>
      </ReplyEvent>
    </transition>
  </transitions>
</ExpectedSystemOperationModel>
```

From an evidence sufficiency point of view, the above model determines the number of events of different types that should be seen whilst monitoring C for the correct realisation of the NR protocol in order to deem the monitoring evidence as sufficient.

5.2.1.5.2 EXPIRATION CONDITION

The second sub-element in the *AssessmentScheme* is the Expiration Condition, which is used to define when a certificate should expire. An example of an expiration condition is given below. The example states that the certificate should expire one year after the date it is issued:

```
<ExpirationCondition Id="987">
   <elapsedPeriod period="1" periodUnit="years"/>
</ExpirationCondition>
```

5.2.1.5.3 ANOMALIES

In this sub-section the anomalies sub-element is presented, with an example of an anomaly for the NR property. Anomalies can be:

- Potential attacks on TOC,
- Other suspicious behaviour in the ToC, or
- Operational conditions related to the security property that is to be certified,

Anomalies are defined as assertions, which despite not having caused any violation of the security property, they may potentially affect its satisfiability and, therefore, lead to the suspension or revocation of certificate generated by the model. The definition of the potential anomalies that should be monitored, as part of a certification model, should be based on an analysis of whether potential attacks, the ways in which the behaviour of different external actors that interact with TOC, and the overall operating conditions of the interaction between TOC and these actors may affect the satisfaction of the given security property by the TOC. Like security

properties, anomalies are also specified as EC-Assertions, except that their violation does not lead automatically to the suspension/revocation of a certificate.

Below we provide an example of defining an assertion for the anomaly concerning Potential attacks. In the case of NR, data owners (As) and data consumers (Bs) may be non-trusted parties. Both of them, for instance, may try to launch a denial-of- service attack on the cloud provider (C). This may happen directly by, for example, issuing a high volume of data uploading and downloading requests to C and/or re-issuing previous requests (replay attack). It should be noted that the monitoring rule R1 in the certification model would require C to respond only to a request from a data provider only if this request has not been responded before. Hence, the certification model assumes that C should not respond to repeated requests. However, even if no response of C is expected in such cases, high volume of repeated requests may escalate to a DOS attack that will prevent C from satisfy the NR property.

Hence the purpose of anomaly monitoring is not to detect the individual instances of repeated requests from A to C, but to detect whether an activity appears in high volume. To monitor and keep a record of the repeated requests from particular data owners, the NR certification model should include the following anomaly detection monitoring assumptions:

```
ANOMALY ASSUMPTIONS:
A1: Initially(RepeatedUplReq(_seqrq, 0), systime())

A2: Happens(e(_id1,_sender1,_receiver1,RQS_AC(_seqrq),_receiver1), _t1,[
_t1,_t1]) ∧ HoldsAt(ResUpReq(_seqrq,_seqrs, _t), _t1) ⇒
Terminates(e(_id1,_sender1,_receiver1,RQS_AC,_receiver1),
RepeatedUplReq(_seqrq, _N), _t1)
∧ Initiates(e(_id1,_sender1,_receiver1,RQS_AC,_receiver1),
RepeatedUplReq(_seqrq,_N+1), _t1)
```

The first of these assumptions initialises the counter of repeated requests from a given data owner _sender1 to 0 and the second increases it whenever a new previously responded requests is re-played by _sender1.

To cover the potential of a similar type of attack from data consumers (B), the NR certification model includes also anomaly-monitoring assumptions similar to those listed above for data downloading requests RQS_{BC}.

The XML example of the assertion for the anomaly is given in the Appendix A denoted with the *Guaranteed ID="gt7"*, based on the EC-Assertion explained. In order to initialise the fluent *RepeatedUpReq*, the *initially* element is used in the postcondition of the first guaranteed. Moreover, in order to increase the variable *_N* of the fluent when it is initialised in the second assumption with the *Guaranteed ID="gt8"*, (in Appendix A) the *operationCall* element is used, which adds the constant *Counter* with value 1 to the variable *_N* of the fluent.

5.2.1.6 MONITORINGCONFIGURATION ELEMENT

The *MonitoringConfigurations* element is used to specify the list of the monitoring configurations that are used to collect the evidence for generating certificates of this type. As shown in the example below, a *MonitoringConfiguration* with *Id=MC1* has been defined. Furthermore, one *Reasoner* called *everestReasoner* has been defined, which is the monitor used for analysing events and checking whether the monitoring conditions are satisfied or not, along with its *EndPoint* and its *assertionID*.

```
<MonitoringConfigurations>
  <MonitoringConfiguration Id="MC1">
    <Component>
      <Reasoner>
        <EndPoint>everestReasoner</EndPoint>
        <AssertionId>AS001</AssertionId>
       </Reasoner>
    </Component>
  </MonitoringConfiguration>
</MonitoringConfigurations>
```

5.2.1.7 EVIDENCEAGGREGATION ELEMENT

An example of an Evidence Aggregation element is shown below. In this example, an aggregation element with start date "2013-01-01" is defined, that states that the aggregation of the detailed evidence, received by the monitor, should be carried out at regular interval periods of 30 days. Therefore, it means that every 30 days the evidence in the generated certificate will be

updated, and should apply a “Max” mathematical function value of the variable used to measure the current security property.

```
<EvidenceAggregation>
  <AggregatedResultsInfo intervalUnit="days" intervalsTime="30"
                         Timestamp="2014-07-01"
                         Startdate="2014-06-01"/>
  <FunctionalAggregatorId>Max</FunctionalAggregatorId>
  <IntermediateResults>false</IntermediateResults>
</EvidenceAggregation>
```

5.2.1.8 LIFECYCLEMODEL ELEMENT

The life cycle model of a monitoring-based certificate is described by the A UML stare chart diagram in the below figure, where all states and transactions are presented.

The life-cycle model of the NR certification model is shown in Figure 48 has an initial state called “*Activated*” and the states “*Pre-Issued*”, “*Issued*”, “*Anomaly Inspection*”, “*Anomaly Selection*”, “*Revoked*” and “*Ended*” (i.e., the final state). Moreover, there are three composite states named “*Continuous Monitoring*”, “*Anomaly-Audit*” and “*Conflict-Audit*”, as well as the historical state “*History*”.

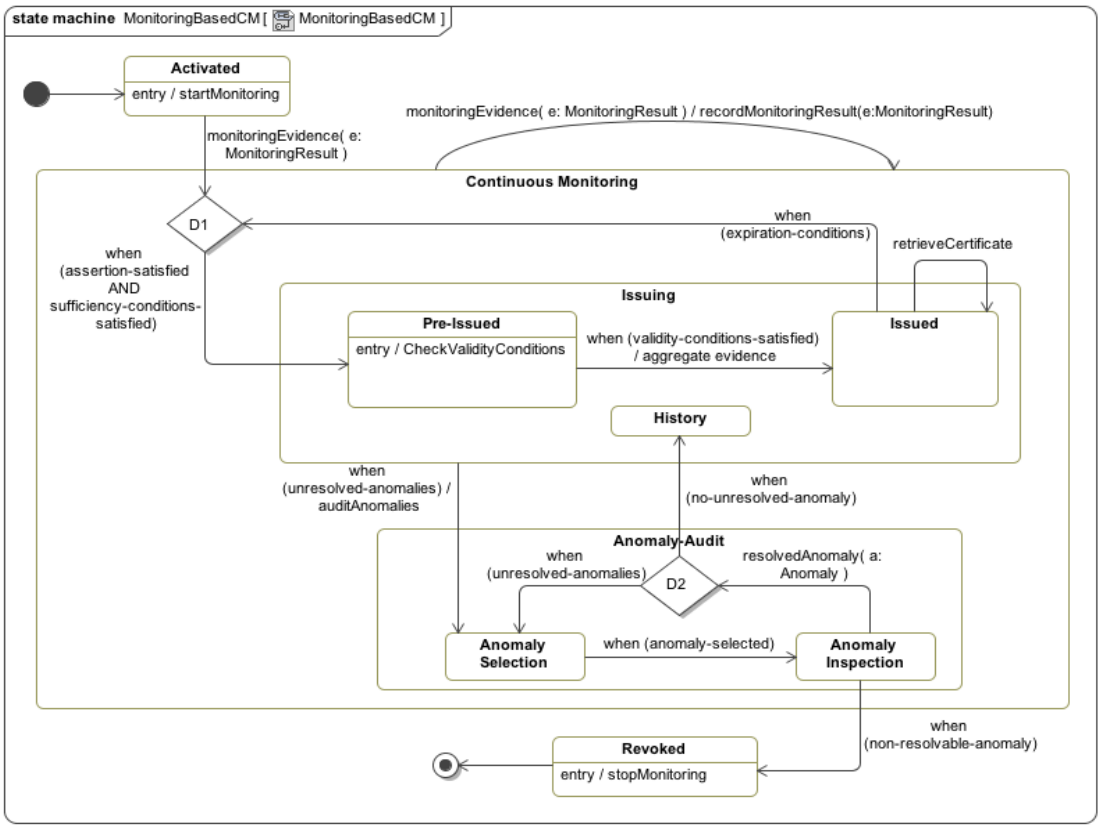


Figure 48 – UML diagram of Life Cycle Model

According to this model, the first state in the certificate's lifecycle is called *Activated*, where the certification process activated and the certificate is being activated. After being activated, the certificate moves to the composite state *Continuous Monitoring*. Whilst being in this state, the security property and anomaly detection monitoring rules and assumptions of the certification model are being monitored by the monitor and the related monitoring evidence is sent to the framework, as stated by the transition *evidence(e:MonResult)*. When the accumulated evidence meets the sufficient conditions (*EvidenceSufficiencyConditions*) specified in the certification model and the security property monitoring rules are satisfied, the certificate moves to the state *Pre-Issued*, according to the transition *when(assertion-satisfied AND sufficiency-conditions-satisfied)* from *D1* to *Pre-Issued* state. At this state, the framework will check if there are any extra validity conditions for the certificate type (see action *CheckValidityConditions*) and, if they

exist and are satisfied, the certificate will move to the state *Issued*, where also the aggregation of the evidence takes place, as specified by the action *aggregate evidence* of the transition that lead to this state. Moreover, at this state a concrete certificate for the NR security property of the specific provider is generated and can be obtained by any interested external party upon request, based on the transition *retrieveCertificate*.

Whilst a certificate is in the *Issuing* composite state, if an anomaly is detected, the it will move to the state *Anomaly-Selection* of the composite state *Anomaly-Audit*, based on the transition *when(unresolved-anomalies)*. When all the detected anomalies are selected, the certificate will move to the state *Anomaly-Inspection* in which the selected anomalies are being inspected one by one. The inspection is a responsibility of the certification authority that will sign off the certificates. If all the detected anomalies are resolved, the certificate moves back to the *History* state, which denotes the state where the certificate was prior to moving to the *Anomaly-Audit*. Otherwise, if there are anomalies that cannot be resolved (i.e., accepted as affordable risks), the certificate moves to the state *Revoke*, which will lead to its termination and no further certificates will be issued.

When the expiration date of an issued certificate is reached, as stated in the *ExpirationCondition* of the certification model, the certificate will move to state *D1* based on the transition *when(expiration-conditions)*. At this point if a sufficient body evidence has already been accumulated for issuing a new instance of the certificate, it will move automatically to the state *Issuing*, or otherwise it will continue gathering evidence until a new certificate instance can be issued.

An XML example of the life-cycle model is given below. In this example all the states are being defined according to their type (atomic, composite, history), as well as their transitions. An example of a transition, as shown in the following example is between the state with id “state1” (“Activated”) and the state with id “state2” (“Pre-Issued”). The condition for this transition states that in order to go from the “Activated” state to the “Pre-Issued” state the *EvidenceSufficiencyCondition*, with id “1011” defined in the *AssessmentScheme* element of the model, should be satisfied.

```
<LifeCycleModel>
  <InitialState stateId="is" name="Activated"/>
  <states>
    <state>
      <atomicState stateId="at1" name="Rejected"/>
    </state>
    <state>
      <compositeState stateId="cs1" name="ContinuousMonitoring">
        <substate>
          <compositeState stateId="cs2" name="Issuing">
            <substate>
              <atomicState stateId="as1" name="Pres-Issued"/>
            </substate>
            <substate>
              <atomicState stateId="as2" name="Issued/>
            </substate>
          </compositeState>
        </substate>
        <substate>
          <atomicState stateId="at2" name="Anomaly-Audit"/>
        </substate>
        <substate>
          <atomicState stateId="at3" name="Conflict-Audit/>
        </substate>
      </compositeState>
    </state>
  </states>
  <transitions>
    <transition From="is" To="at1">
      <WhenCondition>
        <event>assertionViolated</event>
      </WhenCondition>
    </transition>
    <transition From="is" To="as1">
      <WhenCondition>
        <event>sufficiencyConditionSatisfied</event>
      </WhenCondition>
    </transition>
    <transition From="as1" To="as2">
      <WhenCondition>
        <event>assertionSatisfied</event>
      </WhenCondition>
    </transition>
    <transition From="cs1" To="at1">
      <WhenCondition>
        <event>assertionViolated</event>
      </WhenCondition>
    </transition>
    <transition From="cs1" To="fs">
      <WhenCondition>
```

```
        <event>expirationReached</event>
      </WhenCondition>
    </transition>
    <transition From="cs2" To="at2">
      <WhenCondition>
        <event>anomalyDetected</event>
      </WhenCondition>
    </transition>
    <transition From="cs2" To="at3" >
      <WhenCondition>
        <event>conflictDetected</event>
      </WhenCondition>
    </transition>
    <transition From="at2" To="hs">
      <WhenCondition>
        <event>anomalyResolved</event>
      </WhenCondition>
    </transition>
    <transition From="at3" To="hs">
      <WhenCondition>
        <event>conflictResolved</event>
      </WhenCondition>
    </transition>
  </transitions>
  <FinalState stateId="fs" name="Revoked" />
  <historyState stateId="hs" name="history"
                refersToStateId="cs2" />
</LifeCycleModel>
```

5.3 E-HEALTH SCENARIO

The second example is a certification model for an e-Health application developed by Atos[4] that was used as a case study in the CUMULUS project [71]. This application has been built for patients suffering from dementia and it aims to develop an innovative and integrated solution for the general management of such patients and to provide innovative tools to support their treatment. The e-health system has been re-designed and adapted for deployment in a multi-cloud environment, so that it can benefit from the advantages that cloud computing offers. The application consists of two main software components: (i) a multi-cloud server, and ii) a client.

[4] Atos Spain S.A is an international information technology services company that works with clients across the following market sectors: Manufacturing, Retail, Services; Public, Health & Transport; Financial Services; Telecoms, Media & Technology; Energy & Utilities. (http://atos.net)

The first one consists of a database and two server applications. All of them can be deployed in different multi-cloud scenarios alternatives, like private clouds, public clouds or hybrid scenarios. Secondly, the client-side component consists of a desktop application that connects to one of the server applications.

The security property to certify for this system was an Authentication property regarding the HTTP to HTTPS redirection. This security property refers to the network-authenticated-server-access abstract security property of the Application and Interface Security property category (AIS:authentication:network-authenticated-server-access), which guarantees that the exchange of data between the client and the TOC is secure, by verifying the authenticity of both user and TOC.

5.3.1 AUTHENTICATION CERTIFICATION MODEL

As presented in the previous Section 5.2.1, most of the elements that should be defined in the CM can be expressed in a simple way and are usually similar. The different element though in each case is the definition of the assertion of the security property, thus we will provide the EC-Assertion for the network authenticated server access security property used in this scenario.

5.3.1.1 GUARANTEED SUB-ELEMENT

HTTPS to HTTPs redirection states that the TOC provides a communication channel between a client and itself, which offers the following guarantees:

- The client receives assurance that he is communicating with the ToC, and
- The exchange of data between the client and the ToC is protected with guarantees about their authenticity.

This security property has one performance attribute called *verified,* which is a Boolean value to describe if it is true or false, without any parametric attributes. Thus, the property is considered

as verified if the client communicates with a webserver (ToC) using SSL/TLS and verifies the authenticity of the webserver with a certificate, providing adequate encryption. This security property guarantees that HTTP requests are always redirected to HTTPS and that the network channel is confidential, verifying the authenticity of the user and the web server. More specifically, every time there is a request to port 80 (unsecure communication), this request will be redirected to port 443 (secure communication), in order to guarantee redirection. The security property analysed will also verify the confidentiality of the communication, by monitoring the requests sent from the client to the webserver and the requests sent from the webserver to the client, to check if the data is encrypted.

HTTPS (HTTP over SSL or HTTP Secure) is the use of Secure Socket Layer (SSL) or Transport Layer Security (TLS) as a sub-layer under regular HTTP application layering. HTTPS encrypts and decrypts users' page requests as well as the pages that are returned by the Web server. The use of HTTPS protects against eavesdropping and man-in-the-middle attacks. HTTPS and SSL support the use of X.509 digital certificates from the server so that, if necessary, a user can authenticate the sender. Unless a different port is specified, HTTPS uses port 443 instead of HTTP port 80 in its interactions with the lower layer, TCP/IP. The security of HTTPS is therefore that of the underlying TLS, which uses long-term public and secret keys to exchange a short-term session key, in order to encrypt the data flow between client and server.

Therefore, to assess this property, we defined two monitoring rules to cover the cases: i) that the ToC redirects every HTTP request to HTTPS, and ii) that the ToC provides a secure communication (SSL/TLS connection) between the client and the server of the communication. The SSL/TLS encryption validation will verify the presence of SSL/TSL channel encryption.

Concerning the first case of HTTPS redirection, the following monitoring rule applies:

```
Rule 1:
Happens(e(_id1,_sender1,_receiver1,httpCall(_format1,_serverAddress1),
_receiver1),_t1,[_t1,_t1])
∧ _serverAddress1 = 80 ⇒
∧ Happens(e(_id2, _receiver1, _sender1, httpCall(_httpStatus2,
_location2, _serverAddress2), _receiver1),_t2, [_t1, _t1+1000)
∧ _location2 = https://servername/
∧ _httpStatus2 = 301
```

This rule states that when a call to the operation is made from the client to the HTTP server, and the request for the data transmission is through the port 80 (which indicates an unsecure communication channel), the monitor should check for the existence of another event, which is reply event from the web server back to the client, in order to notify the execution of the redirection to the HTTPS port. For every redirection, the reply event should have an attribute named *location*, which will be a string that indicates that there is redirection to the HTTPS port, and an *HttpStatus* attribute, which includes the status code of the response, that should be 301.

The Guaranteed element for the above rule is presented in the XML listing below:

```
<Guaranteed ID="gt1" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <quantification>
    <quantifier>existential</quantifier>
    <timeVariable>
      <varName>t2</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="ac0">
      <eventCondition unconstrained="true">
        <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>serverHTTP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>httpCall</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
              <value>REQ-B</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
```

```
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>format1</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serverAddress1</varName>
              <varType>string</varType>
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </fromTime>
          <toTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
    <WrappedCondition>
      <operator>and</operator>
      <assertionCondition>
        <atomicCondition>
          <relationalCondition>
            <equal>
              <operand1>
                <variable>
```

```
                <varName>serverAddress1</varName>
                <varType>string</varType>
              </variable>
            </operand1>
            <operand2>
              <constant>
                <name>port80</name>
                <value>80</value>
              </constant>
            </operand2>
          </equal>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </relationalCondition>
      </atomicCondition>
    </assertionCondition>
  </WrappedCondition>
</precondition>
<postcondition>
  <atomicCondition conditionID="ac1">
    <eventCondition unconstrained="true">
      <event>
        <eventID forMatching="true" persistent="false">
          <varName>VID1</varName>
        </eventID>
        <reply>
          <interfaceId>serverHTTP</interfaceId>
          <OperationId>2</OperationId>
          <operationName>httpCall</operationName>
          <outputVariable forMatching="true" persistent="false">
            <varName>status2</varName>
            <varType>OpStatus</varType>
            <value>RES-B</value>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
```

```
            <varType>string</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>httpStatus2</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>location2</varName>
            <varType>string</varType>
          </outputVariable>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>serverAddress2</varName>
            <varType>string</varType>
          </outputVariable>
        </reply>
        <tVar>
          <timeVar>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
          <Expression>
            <plus>1000</plus>
          </Expression>
        </toTime>
      </event>
    </eventCondition>
  </atomicCondition>
  <WrappedCondition>
    <operator>and</operator>
    <assertionCondition>
      <atomicCondition>
        <relationalCondition>
          <equal>
            <operand1>
              <variable>
                <varName>location2</varName>
                <varType>string</varType>
```

```
              </variable>
            </operand1>
            <operand2>
              <constant>
                <name>locationHttps</name>
                <value>https://servername/</value>
              </constant>
            </operand2>
          </equal>
          <timeVar>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </relationalCondition>
      </atomicCondition>
    </assertionCondition>
  </WrappedCondition>
  <WrappedCondition>
    <operator>and</operator>
    <assertionCondition>
      <atomicCondition>
        <relationalCondition>
          <equal>
            <operand1>
              <variable>
                <varName>httpsStatus2</varName>
                <varType>string</varType>
              </variable>
            </operand1>
            <operand2>
              <constant>
                <name>RedirectStatus</name>
                <value>301</value>
              </constant>
            </operand2>
          </equal>
          <timeVar>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </relationalCondition>
      </atomicCondition>
    </assertionCondition>
  </WrappedCondition>
 </postcondition>
</Guaranteed>
```

In order to check that the ToC provides a secure communication (SSL/TLS connection) between the client and the server of the communication, the following rule applies.

```
Rule 2:
Happens(e(_id1,_sender1,_receiver1,httpCall(_format3,_serverAddress4),
_receiver1),_t3,[_t3,_t3])
∧ _serverAddress4 = 443 ⇒
_format3 = SSL-TLS
```

This rule states that when a call event to the *HttpsCall* operation is made from the HTTPS server to the client, and the data transmission through port 443, then the monitor should check if the value of the argument *format*, which describes the content of the packet, is *SSL-TLS*. The XML representation of this rule can be found in Appendix B, with the *GuaranteedID="gt2"*.

5.4 SMART CITIES SCENARIO

The Smart Cities scenario was developed by Wellness Telecom[5] and was used as a case study in CUMULUS project [70]. This scenario was developed in order to manage the public lighting in Spain in a more effective and efficient way. *WeLight* was developed to achieve significant energy and economic savings on public lightning, and to provide real-time information about the performance of the system and the status of the public infrastructure. There are several standard interfaces available that can be used to interact with several devices that may exist within an electrical panel, such as network analysers, flux regulators, contactors or peer-to-peer telemanagement systems. One of the strengths of *WeLight* is the fact that it does not need its own server or its own communications infrastructure, as all the information is available through an Internet-based software service. It is a SaaS solution based on VMWare technology, which is used on top of Open Stack for IaaS.

[5] Wellness Telecom (WT) is an Information and Telecommunication Technologies SME company located in Seville (Spain). WT is expert in wireless/sensors networks management, open source software adapted to ICT solutions, and development of intelligent control systems able to interoperate with multiple technologies. (www.wtelecom.es/)

The security property that is the focus of certification in this system scenario is data access confidentiality between two parties. This is an example of the external data exchange confidentiality security property (*AIS:confidentiality:external-data-exchange-confidentiality*).This property is achieved in the scenario system through the use of a VPN (Virtual Private Network). VPN is a private network that allows data to securely pass form one endpoint to the other, thus it provides additional security by hiding the traffic from the public network that the service operates.

5.4.1 CONFIDENTIALITY CERTIFICATION MODEL

The different elements that define the confidentiality certification model are introduced in the reminder of this chapter.

5.4.1.1 GUARANTEED SUB-ELEMENT

The external data exchange confidentiality security property describes the confidentiality level of the data in the TOC with respect to the personnel operating the TOC. Thus, data should only be accessible within the TOC and no external entities should have access to this data.

To verify this property the monitor has to check if the data exchanges between the involved parties are sent through the VPN. In order to do so, we need the address through which the data is sent. This address is monitored by accessing the appropriate datagram field in the Internet Layer. Moreover, we assume that the events can be captured only on one side of the communication channel, which is the server side. The opposing party (i.e. the client) is assumed to receive what is sent from the server and to have sent what the server receives.

Thus, in order to assess this property, we have defined three rules, which are presented below in EC-Assertion.

```
Rule 1:
Happens(e(_id1,_sender1,_receiver1,LCUCall(_packetDest),_receiver1),
_t1,[_t1,_t1])
∧ HoldsAt(vVPNaddress(_VPNaddressCall,_t), _t1) ⇒
_packetDest = VPNaddressCall
```

The first rule states that when data is sent from the server to the client, the datagram containing the data is sent to the address of the VPN. The guaranteed term to express this rule is given below.

```
<Guaranteed ID="gt1" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="ac0">
      <eventCondition unconstrained="true">
       <event>
         <eventID forMatching="true" persistent="false">
           <varName>VID0</varName>
         </eventID>
         <call>
           <interfaceId>LCU</interfaceId>
           <OperationId>1</OperationId>
           <operationName>LCUCall</operationName>
           <inputVariable forMatching="true" persistent="false">
             <varName>status1</varName>
             <varType>OpStatus</varType>
           </inputVariable>
           <inputVariable forMatching="true" persistent="false">
             <varName>sender1</varName>
             <varType>Entity</varType>
           </inputVariable>
           <inputVariable forMatching="true" persistent="false">
             <varName>receiver1</varName>
             <varType>Entity</varType>
           </inputVariable>
           <inputVariable forMatching="true" persistent="false">
             <varName>source1</varName>
             <varType>Entity</varType>
           </inputVariable>
           <inputVariable forMatching="true" persistent="false">
```

```
            <varName>serviceId</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>packetDest</varName>
            <varType>string</varType>
          </inputVariable>
        </call>
        <tVar>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </toTime>
      </event>
    </eventCondition>
  </atomicCondition>
  <WrappedCondition>
    <operator>and</operator>
    <assertionCondition>
      <atomicCondition conditionID="as1">
        <stateCondition>
          <holdsAt>
            <state name="vVPNaddressCall">
              <argument>
                <variable forMatching="true" persistent="false">
                  <varName>VPNaddress</varName>
                  <varType>string</varType>
                </variable>
              </argument>
            </state>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </holdsAt>
        </stateCondition>
      </atomicCondition>
```

```
      </assertionCondition>
    </WrappedCondition>
  </precondition>
  <postcondition>
    <atomicCondition conditionID="as2">
      <relationalCondition>
        <equal>
          <operand1>
            <variable forMatching="true" persistent="false">
              <varName>packetDest</varName>
              <varType>string</varType>
            </variable>
          </operand1>
          <operand2>
            <variable forMatching="true" persistent="false">
              <varName>VPNaddressCall</varName>
              <varType>string</varType>
            </variable>
          </operand2>
        </equal>
        <timeVar>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVar>
      </relationalCondition>
    </atomicCondition>
  </postcondition>
</Guaranteed>
```

```
Rule 2:
Happens(e(_id1,_sender1,_receiver1,LCUCall(_packetSource),_receiver1),
_t1,[_t1,_t1])
∧ HoldsAt(vVPNaddress(_VPNaddress,_t), _t1) ⇒
_packetSource = VPNaddress
```

The second rule states that when the server receives data, the datagram containing the data is received from the address of the VPN.

However, the VPN address required for the above rules is set by calls to a particular operation (*setupVPN*), if the VPN server is determined dynamically. The rule about initializing the VPN address in the formulas is expressed in EC-Assertion as:

```
Rule 3:
Happens(e(_id1,_sender1,_receiver1,setupVPN(_setupVPNaddress),
_receiver1),_t1,[_t1,_t1])
∧ HoldsAt(vVPNaddress(_VPNaddress,_t), _t1) ⇒
Terminates(e(_id1,_sender1,_receiver1,setupVPN(_setupVPNaddress),
_receiver1), vVPNaddress(_VPNaddress,_t), _t1)
∧ Initiates(e(_id1,_sender1,_receiver1,setupVPN(_setupVPNaddress),
_receiver1), vVPNaddress(_setupVPNaddress,_t), _t1)
```

The Guaranteed terms of the last two rules expressed in XML can be found in Appendix C, with the *GuaranteedID "gt2"* and *"gt3"*, respectively. The rest of the elements are of the certification model are similar to those presented for the NR security property, except of the Anomalies element, which is not relevant for the last two examples of the scenarios provided.

Chapter Six

IMPLEMENTATION

6.1 OVERVIEW

In this chapter we discuss the implementation of the proposed framework for generating monitoring-based certificates, as described in Chapter 4. More specifically in section 6.2 is presented the implementation architecture of the framework and the different components that it consists of. Section 6.3 presents a more detailed explanation of the components along with their implementation and their interfaces (APIs). Finally, also the databases used are also presented.

6.2 ARCHITECTURE

The monitoring mechanisms of this research have been designed according to the architecture model shown in Figure 49. This figure presents the internal design of the framework for realising the generation of monitoring-based certificates. All monitoring modules and the databases of the framework are presented.

As shown in the figure below, there are three different types of actors, which are:

- **A CA/ Cloud Service Provider,** which is a Certification Authority or a cloud provider that requires to certify a cloud service,
- **A Service Consumer,** which is a user eligible to request a certificate of a specific service (i.e. a privileged service user, a cloud provider, a service owner, etc.), and
- **A Client,** which is any user that uses or interacts with the cloud service that is being monitored and certified.

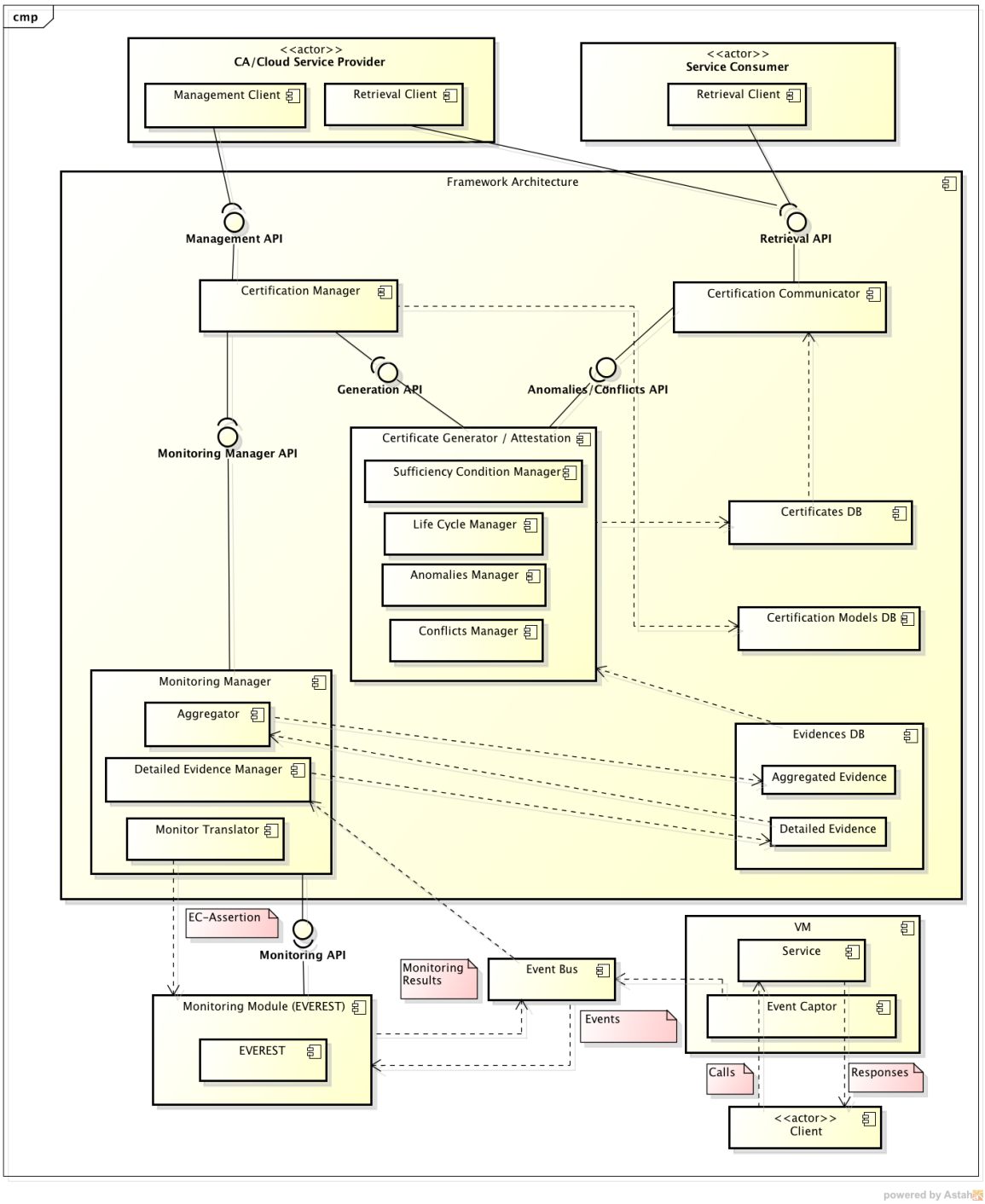


Figure 49 – Monitoring Architecture

Based on the architecture presented in the above figure, the monitoring mechanisms of the proposed framework are realised through the following components:

- **Certification Manager:** This component communicates with actors to create or modify Certification Models (CMs), and then it provides the information of them to the Monitoring Manager, to start the certification process.

- **Certification Communicator:** This component communicates with actors that request a certificate and send them the generated certificates, if there are any generated ones from the required CM stored in the *Certificates Repository*.

- **Certificate Generator/Attestation:** The Certificate Generator Attestation component is the component that has responsibility to check all the conditions stated in the CM and to manage the process of generating certificates according to the life-cycle model defined in the CM. It consists of four sub-components, each of which is responsible for a specific type of conditions stated in the CM. More specifically, these sub-components are:

  - **Anomalies Manager**, which is the component with the responsibility to handle anomalies that might be defined in the CM as assertions,

  - **Conflicts Manager**, which is the component that has the responsibility to handle conflicts that might be defined, in the CM

  - **Sufficient Condition Manager**, which is the component with the responsibility to handle sufficiency conditions, and

  - **Life-Cycle Manager:** This component is responsible to handle the life cycle model stated in the Certification Model and change the status of the certificate based on the conditions and states provided by the state-transition model of the life cycle.

- **Monitoring Manager:** The Monitoring Manager component retrieves the CM instance from the Certification Manager, which holds the information about the *ToC* to be certified, the security property, as well as the conditions that should be taken

under consideration in order to issue a certificate. After retrieving the instance, the Monitoring Manager feeds the other components with the relevant information of the CM instance, depending on their functionality.

More specifically, it receives the certification model from the Certification Manager module and configures the required monitoring infrastructure for the realization of the certification model. It also configures the Detailed Evidence Manager and the Aggregation Manager, which are responsible to process the monitoring results. Finally, it also translates the security property assertions to the language that the monitor understands through the Monitor Translator module.

To fulfil these responsibilities it consists of the following sub-components:

- **Detailed Evidence Manager:** This component polls the monitoring module at regular intervals, in order to collect the monitoring results generated for the given certification model, and stores the monitoring results in the *Detailed Evidence* table of the framework's database.

- **Aggregator:** This component has responsibility for aggregating monitoring results and storing them in the *Evidence DB* of the framework, as required by the given certification model. The aggregation manager polls the detailed evidences at regular intervals, in order to aggregate and store these events in an aggregated form, which is also defined by the certification model. For example, primitive monitoring results may denote lengths of individual periods of unavailability of a service, whereas the aggregated results may indicate the average length of unavailability periods over a monitoring period of a specific period.

- **Monitor Translator**: This component retrieves the information about the assertions of the security property described in the Certification Model, and translates them to the EC-Assertion language that the Monitor understands, in order to start the monitoring process of the specific ToC for the defined security property.

- **Monitor Module:** The Monitor Module is responsible for monitoring the events for a specific security property and a target of certification (ToC). This module may reside on a cloud infrastructure or can be external. In our current implementation, we assume that it is an external module. The Monitor Module gets a monitoring configuration with all the details required for the monitoring process (e.g., the assertions that constitute the specification of the security properties that need to be monitored at runtime, or where to report the results of the monitoring process) from the Monitoring Manager. The Monitor Module consists the actual monitor that we are using, which is the EVEREST monitoring tool.

- **Event Bus**: This component has responsibility for communicating primitive monitoring events from and monitoring results from the between the different components of the monitoring infrastructure used in the generation of monitoring based certificates, and between such components and our framework.

- **Event Captor:** This component is responsible to capture all the events that are happening in the ToC, in order to monitor a ToC and gather the required evidence for certifying it. It publishes the events that they capture to the allocated Event Bus, which subsequently forwards them to the monitor.

6.3 COMPONENTS

In this section each individual component of the presented monitoring architecture in Figure 49 is explained, as well as the interfaces exposed by the different components in the architecture.

6.3.1 CERTIFICATION MANAGER

This component communicates with external actors, in order to create or modify an existing Certification Model (CM). Moreover, it provides the information of the created CM to the Monitoring Manager, to start the certification process, as presented in Figure 50.

The Certification Manager component is responsible to handle requests from Certification Authorities (CA) or Cloud Service Providers about generating Certification Models or issuing new certificates for specific security properties. After receiving the requests, the Certification Manager will pass all the necessary information of the agreed CM to the Monitoring Manager Module to begin the certification process, through the *Monitoring Manager API*.

This component is also connected with the Certification Models DBs, so as to retrieve all necessary information about the required models from the CA. Moreover, it provides information stored in the Certification Model to the Certification Generator/Attestation module through the *Generation API*, for the generation of new certificates.

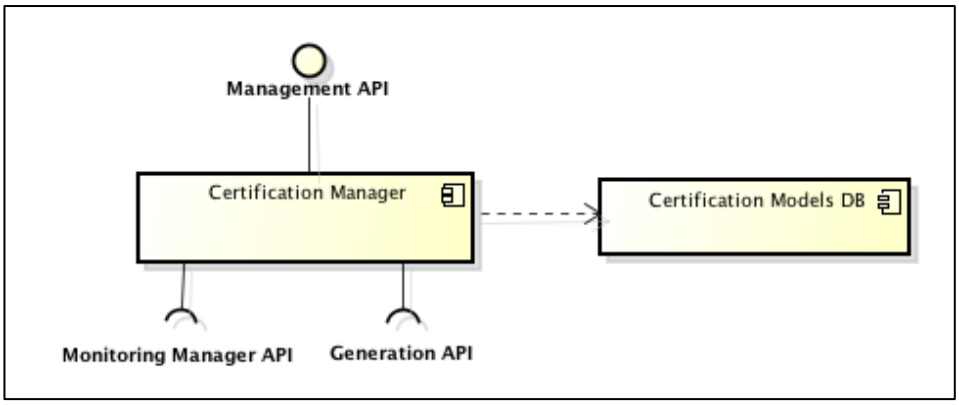


Figure 50 – Certification Manager

The specifications and the operations of the interface provided by this component are presented in the Table 10 below.

Table 10 – Management API

Management API	
Operation	**Description**
`getPropertyAndTOCS(): String`	This method receives a call from the CA to retrieve all available TOCz and Security properties, from the available certification models stored in the database.

Return Type	String	This method returns a string representation of an XML of the following structure, which describes all the available properties and TOCs.
		`<PropertiesAndTocs>`
		`<Property category="xxxxxx">`
		`<TargetOfCertification ID="yyyyy" />`
		`</Property>`
		`<Property category="xxxxxxx">`
		`<TargetOfCertification ID="yyyyy" />`
		`</Property>`
		`… … …`
		`… … …`
		`</PropertiesAndTocs>`

Operation	**Description**
`retrieveCertificationModel(String property, String toc): String`	Retrieves all available certification models for the specific TOC and property chosen from the CA.

Parameters

Name	**Type**	**Description**
Property	String	The selected property by the CA to be certified
TOC	String	The selected TOC by the CA
Return Type	String	This method returns the string representation of the XML certification model, where the XML is written according to the schema presented in Chapter 4.

Operation	**Description**
`addCertificationModel(String cmInstanceId, String modelXML)`	This method stores the confirmed certification model of the CA to the database.

Parameters

Name	**Type**	**Description**
cmInstanceId	String	The unique identifier of the certification

		model instance that has been confirmed
modelXML	String	The confirmed CM in the XML format

Operation		**Description**
`submitCertificationModel (String modelXML)`		This method allows the CA to submit a certification model in order to start the certification process.
Parameters		
Name	**Type**	**Description**
modelXML	String	The string representation of the XML certification model.

6.3.2 MONITORING MANAGER

The Monitoring Manager component presented in Figure 51 is responsible for coordinating the automatic configuration of the monitoring system and for managing the monitoring process required for certifying a specific security property. More specifically, when it receives the necessary information of the provided certification model (CM) from the Certification Manager, its sub-component Monitor Translator translates the assertion defined in XML, into EC-Assertion, which is the language that the Monitor understands to start monitoring the events of the specific ToC. In order to receive a CM from the Certification Manager component, the Monitoring Manager exposes the Monitoring Manager API, which is presented in Table 11.

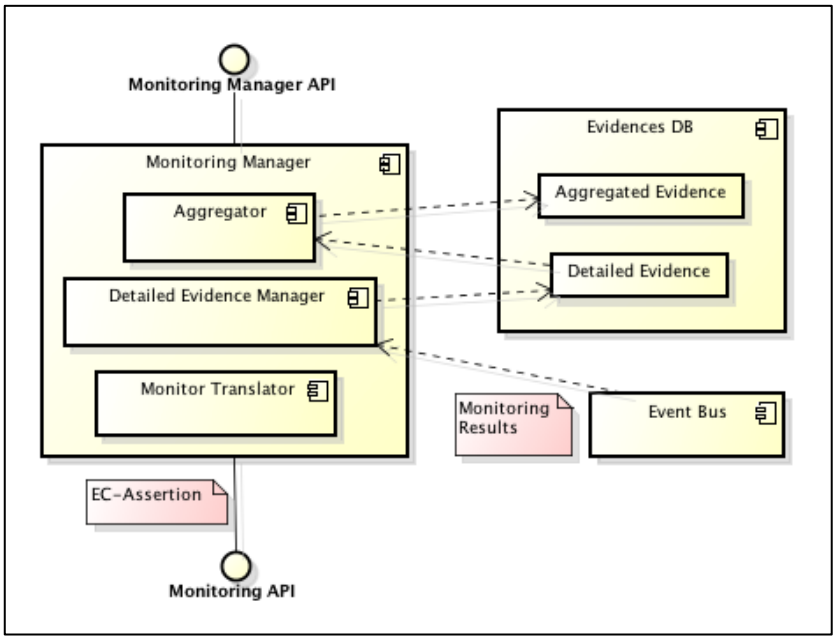


Figure 51 – Monitoring Manager

Table 11 – Monitoring Manager API

Monitoring Manager API		
Operation	**Description**	
`submitCertificationModel (String model)`	This method allows submitting a certification model to the monitoring manager from the Certification Manager.	
Parameters		
Name	**Type**	**Description**
model	String	String representation of an XML certification model.

6.3.2.1 DETAILED EVIDENCE MANAGER

This component is part of the Monitoring Manager component and is responsible to poll the monitoring module at regular intervals, in order to collect the produced monitoring results. If it fetches any monitoring result from the monitoring module, it stores the results in the "Detailed Evidence" table of the framework database. This evidence is called *Detailed* (and consequently the component that manages it is called *"Detailed Evidence Manager"*), as it refers to every single result coming from the monitor, as opposed to the "Aggregated Evidence", which is the aggregated value of more than one monitoring results.

6.3.2.2 AGGREGATION MANAGER

Aggregation Manager is a part of the Monitoring Manager component and is responsible to aggregate the monitoring results. The monitoring manager configures the aggregation manager in order to aggregate the monitoring results for every aggregated period, as defined in the certification model. The Aggregation Manager component also stores the aggregated results in the "Aggregated Evidence" table of the framework database.

6.3.2.3 MONITOR TRANSLATOR

Following the introduction of a new language for specifying assertions as part of monitoring based certification models, it became necessary to introduce a new component that would support the translation of assertions into the operational monitoring language of the monitor used, which is the EVEREST monitoring tool. In particular, the new component is responsible to translate the assertions for a security property as specified in the CM, into EC-Assertion. EC-Assertion is an XML language based on Event Calculus and is expressed we have discussed already in Chapter 3.

This new assertion language for monitoring based certification models has a similar semantic foundation to EC-Assertion (i.e., it is based on the notion of the events that may happen at

runtime and affect the state of a TOC) but it provides higher level and aggregate syntactic constructs in order to specify security property assertions. The following table provides an overview of the correspondences between the CM assertion specification elements and the EC-Assertion.

Table 12 - Correspondences between CM assertion specification elements and EC-Assertion

CM Assertions	EC-Assertions	Comments
ID	Id of formulas	Assertions in a CM must have identifiers that uniquely identify them within the CM. EC-Assertion formulas are also required to have a unique id within a monitoring specification.
variables (operation vars, temporal, vars, state condition vars and relational condition vars)	Variables (operation vars, temporal, vars, fluent vars and relational condition vars)	There is a 1-1 correspondence between operational, temporal and relational conditions variables in CM assertions and EC-Assertions. State condition variables in CM assertions correspond to fluent variables in EC-Assertion.
quantifiers (forall, exists)	quantifiers (forall, exists)	Both CM assertions and EC-Assertion formulas have variables quantified by either the universal (forall) or the existential (exists) quantifier.
logical operators (and, or, not)	logical operators (and, or, not)	Both languages support the use of the logical operations of conjunction (and), disjunction (or) and negation (not) to form compound logical conditions within assertions (CM assertion) and formulas (EC-Assertion).
precondition	*body* of monitoring formulas	Precondition element of the CM assertion corresponds to the body in EC-Assertion language.
postcondition	*head* of monitoring formulas	Postcondition element of the CM assertions corresponds to the head in EC-Assertion language
event condition	*Happens* predicate	Event conditions in CM assertions correspond to conditions expressed by the predicate Happens in EC-Assertion. Call,

		reply, and execute events in CM-Assertion correspond to REQ, RES and EXC events in Happens predicates. In both languages these types of events relate to the execution of operations. A time variable (timeVar) in a CM assertion event corresponds to the time variable of a *Happens* predicate. A time period (*timePeriod*) in a CM assertion event has no corresponding counterpart in EC-Assertion and was introduced for this purpose. The range of a time variable in a CM assertion event corresponds the time range of a *Happens* predicate. As in CM assertions, the boundaries of a time range in EC-Assertion can be defined by time variables, constants, or linear time expressions over time variables and constants.
time expression	time expression	Both CM assertions and EC-Assertions support the specification of linear time expressions over time variables and constants.
operation Type	signature (sig)	EC-Assertion supports the specification of operations but does not group them into interfaces. Furthermore, EC-Assertion does not distinguish between input, output and fault variables in operations. Hence, all these different types of operation variables in CM assertions are mapped into variables in EC-Assertion.
state condition – initiates element	*Initiates* predicate	State conditions in CM assertions are mapped into fluents in EC-Assertion. Initiates conditions in CM assertions are expressed by an Initiates predicate in EC-Assertion, which expresses the initiation of a fluent that represents the condition at a specific time point.
state condition – terminates element	*Terminates* predicate	State conditions in CM assertions are mapped into fluents in EC-Assertion. Terminates conditions in CM assertions are expressed by a Terminates predicate, which expresses the termination of a fluent that represents the condition at a specific time point.

state condition – initially element	*Initially* predicate	State conditions in CM assertions are mapped into fluents in EC-Assertion. An initially state condition in CM assertions is expressed by an Initially predicate, which expresses the holding of the fluent that represents the condition at the start of the system's operation.
state condition – holdsAt element	*HoldsAt* predicate	State conditions in CM assertions are mapped into fluents in EC-Assertion. A holdsAt state condition in CM assertions is expressed by a HoldsAt predicate, which expresses the holding of the fluent that represents the condition at a particular time point.
relational conditions	*relational* conditions	Relational conditions in CM assertions are mapped directly into relational predicates in EC-Assertion. The only difference is that a relational condition may involve an event series expression as an operand of a function that is an operand of a relational condition. In this case the mapping is described below.
relational condition operand (variable, operation call, expression, event series expression)	relational condition operand (variable, operation call, expression)	Operands in relational conditions of both languages correspond to each other with the exception of event series expressions in CM assertions.
event series expression		Event series expressions in CM assertions have no direct corresponding element in EC-Assertion language. Event series expressions are mapped into a set of monitoring assumptions, which are used to detect the event patterns of the event series expression, and update fluents keeping a record of intermediate values of the overall variable of the event series expression. The mapping of a CM assertion event series expression involve also a monitoring rule that checks the condition expressed by the relational condition, which incorporates the event series expression.

Based on the above correspondences, we have developed an algorithm that translates CM assertions into EC-Assertion formulas. The specification of this algorithm is given in the following table.

As shown in the table, the algorithm is a collection of functions that collectively transform CM assertions into EC-Assertion formulas. More specifically, the function *translateAssertionToEC* accepts a CM assertion and returns a list of EC formulas. This function creates an EC-Assertion formula for the given CM assertion and then invokes the *translateAssertionCondition* function to transform the precondition and postcondition of the assertion into EC-Assertion formulas predicates that form the body and head respectively of the EC formula. The function *translateAssertionCondition* transforms the atomic conditions. In particular, event conditions are transformed into Happens predicates in EC-Assertion formulas, state conditions are transformed into Initiates, Terminates or *HoldsAt* predicates in EC-Assertions, and relational conditions in CM assertions are transformed into relational predicated in EC-Assertion. These transformations are supported by the functions *createECEventPredicate*, *createECStatePredicate* and *createECRelationalPredicate* in the algorithm. The function *addOperand* transforms the operand of a relational condition into EC relational operands. If a relational operand is of type series expression, then this function also creates additional assumptions to update the computational value of the series expression.

Table 13 - Algorithm for translating CM assertions into EC-Assertions formulas

```
1   Aec[] : List of EC Assumptions

2   Function translateAssertionToEC(AT):Fec[]
3     Fec[] := Φ
4     Aec[] := Φ
5     for each V in AT.V[]do
6       create a fluent FLec to hold V
7       create Initially formula Fec for FLec
8       Fec[] := Fec[] U Fec
9     end for

10    for each AF in AT.AF[]do
11      create empty EC formula Fec
12      Fec.B[] := Fec.B[]UtranslateAssertionCondition(AF.PrC)
13      Fec.H[] := Fec.H[]UtranslateAssertionCondition(AF.PoC)
14      Fec[] := Fec[]U Fec
15    end for
16    Fec[] := Fec[]U Aec[]
```

```
17    return Fec[]
18  end translateAssertionToEC

19  Function translateAssertionCondition(AC):Pec[]
20    Pec[] := Φ
21    for each atomic condition/wrapped atomic condition ATC in AC do
22      if ATC is an event condition then
23        Pec[] := Pec[]UcreateECEventPredicate(ATC)
24      else if ATC is a state condition then
25        Pec[] := Pec[]UcreateECStatePredicate(ATC)
26      else if ATC is a relational condition then
27        Pec[] := Pec[]UcreateECRelationalPredicate(ATC)
28      end if
29    end for
30    return Pec[]
31  end translateAssertionCondition

32  Function createECRelationalPredicate(REL):Pec[]
33    Pec[] := Φ
34    create an EC relational Pec for REL
35    Pec[] := Pec[]UaddOperand(Pec, REL.OP1)
36    Pec[] := Pec[]UaddOperand(Pec, REL.OP2)
37    Pec[] := Pec[] + Pec
38    return Pec[]
39  end createECRelationalPredicate

40  Function addOperand(Pec, OP):Pec[]
41    Pec[] := Φ
42    if OP is a Function operandthen
43      create EC function Fnec for OP
44      add Fnec to Pec as an operand
45    else if OP is a series expression operand then then
46      let SE is the series expression
47      if SE.COM is a function then
48        let Fn is the function
49        Pec[] := Pec[]UtranslateAssertionCondition(SE.AC)
50        Pec[] := Pec[]UadjustFunction(Fn)
51        create an EC function Fnec for Fn
52        create EC variable Vec to hold return value of Fnec
53        create a Fluent FLec to hold Vec
54        add Vec to Pec as an operand
55        create Initially formula Fec for FLec
56        Aec[] := Aec[]U FLec
57        create an assumption Aec to Terminate/Initiate FLec using Pec[] and
58        Fnec
59        Aec[] := Aec[]U Aec
60    else
61      create EC variable Vec for OP
62      add Vec to Pec as an operand
63    end if
```

```
64     return Pec[]
65   end addOperand
```

Symbols	
P_{ec} – EC predicate	AF – Assertion Formula
P_{ec}[] – List of EC predicates	AF.PrC – Precondition of type Assertion Condition
F_{ec} – EC Formula	AF.PoC – Post condition of type Assertion Condition
F_{ec}[] – List of EC Formulas	AC – Assertion Condition
A_{ec}[] – List of EC Assumptions	AC.ATC – Atomic Condition in AC
F_{ec}.B[] – List of P_{ec} signifies formula body	AC.WC[] – List of wrapped condition of type AC
F_{ec}.H[] – List of P_{ec} signifies formula head	REL - Relational Condition
FL_{ec} – EC fluent	REL.OP1 - First operand of the relation
V_{ec} – EC variable	REL.OP2 - Second operand of the relation
AT – Assertion	SE - Series Expression
AT.V[] – List of variable declaration in an assertion	SE.AC - Assertion Condition
AT.AF[] – List of assertion formula in an assertion	SE.COM - Computation

Functions

adjustFunction(Fn):P_{ec}[] - Is a function that checks if the argument type of the series function Fn complies with the expected argument type. If the argument type does not match the expected argument type, it amends the argument to make it to comply with the expected argument type.

createECEventPredicate(ATC):P_{ec}– This function accepts an event atomic condition in an assertion and creates the corresponding EC Happens predicate.

createECStatePredicate(ATC):P_{ec} – This function accepts a state atomic condition in an assertion and creates the corresponding EC predicate (i.e. Initiates, Terminates or HoldsAt).

In order to provide the EC-Assertion formula to the Monitor, the Monitoring Manager uses the Monitoring API, which is presented in Table 14.

Table 14 – Monitoring API

Monitoring API		
Operation		**Description**
`feedEverestWithMonitoringSpecifications (String formula)`		This method is responsible for submitting the assertions for a security property as specified in the CM to the Monitor component.
Parameters		
Name	**Type**	**Description**
formula	String	String representation of the EC-Assertion formula.

6.3.3 CERTIFICATION COMMUNICATOR

This module presented in Figure 52 allows the actors to request a generated Certificate. To enable this communication the Certification Communicator component exposes the Retrieval API. Moreover, in case the framework detects an anomaly or a conflict, it receives this information from the Certificate Generator attestation component through the Anomalies/Conflicts API.

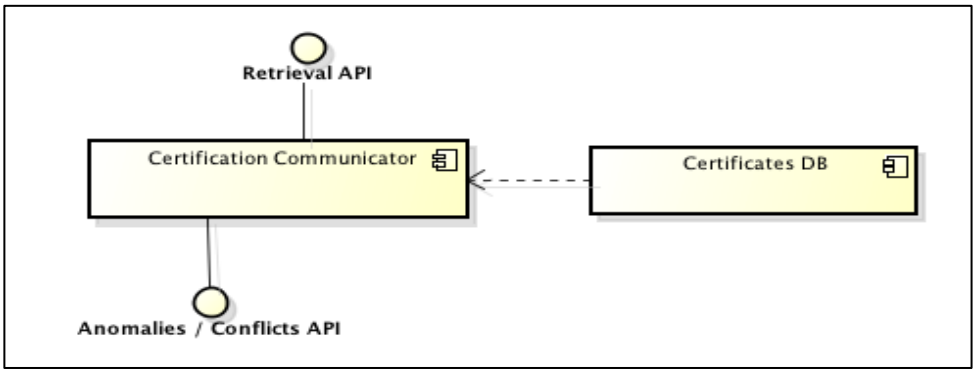


Figure 52 – Certification Communicator

The Retrieval API presented in Table 15 defines methods to establish the communication between the actors and the framework, in order to handle the requests for generating specific certificates.

Table 15 – Retrieval API

Retrieval API		
Operation		**Description**
`Get(String Certificate_Id, String SecProperty, String TOC)`		This method retrieves tall certificates for the specific Security Property and ToC from the Certificates DB.
Parameters		
Name	**Type**	**Description**
Certificate_Id	String	The unique identifier of the certificate
SecProperty	String	The security property that the generated certificate refers to.
TOC	String	The ToC that the generated certificate refers to.

Operation	**Description**
`notifyAnomaliesandConflicts (String certId): Boolean`	This method allows the CA to be notified about a detected anomaly or conflict, regarding a specific certificate.

Parameters		
Name	**Type**	**Description**
certId	String	The Id of the specific generated certificate that the anomaly or a conflict refers to.
Return Type	Boolean	This method returns ta Boolean value regarding the resolution of an anomaly or a conflict. True means that it was resolved and False that it was not resolved.

In order for the Certification Communicator to receive any anomalies or conflicts detected through the monitoring process, in exposes the Anomalies/Conflicts API to communicate with the Certificate Generator Attestation component. The method of this API is presented in the Table 16.

Table 16- Anomalies and Conflicts API

AnomaliesandConflicts API	
Operation	**Description**
`getAnomaliesandConflicts (String certId): Boolean`	This method allows the CA to be notified about a detected anomaly or conflict, regarding a specific certificate.
Parameters	

Name	**Type**	**Description**
certId	String	The Id of the specific generated certificate that the anomaly or a conflict refers to.
Return Type	Boolean	This method returns ta Boolean value regarding the resolution of an anomaly or a conflict. True means that it was resolved and False that it was not resolved.

6.3.4 CERTIFICATE GENERATOR / ATTESTATION

The Certificate Generator/Attestation (CG) is presented in Figure 53. This component has the responsibility to check all the conditions stated in the CM and managing the process of generating certificates according to the life-cycle model defined in the CM. It consists of four different sub-components, each of which is responsible for a specific type of conditions defined in the CM, in order to issue a certificate and update its status. It receives the CM from the Certification Manager component and for this reason it exposes its functionality through the Generation API, which is presented in Table 17.

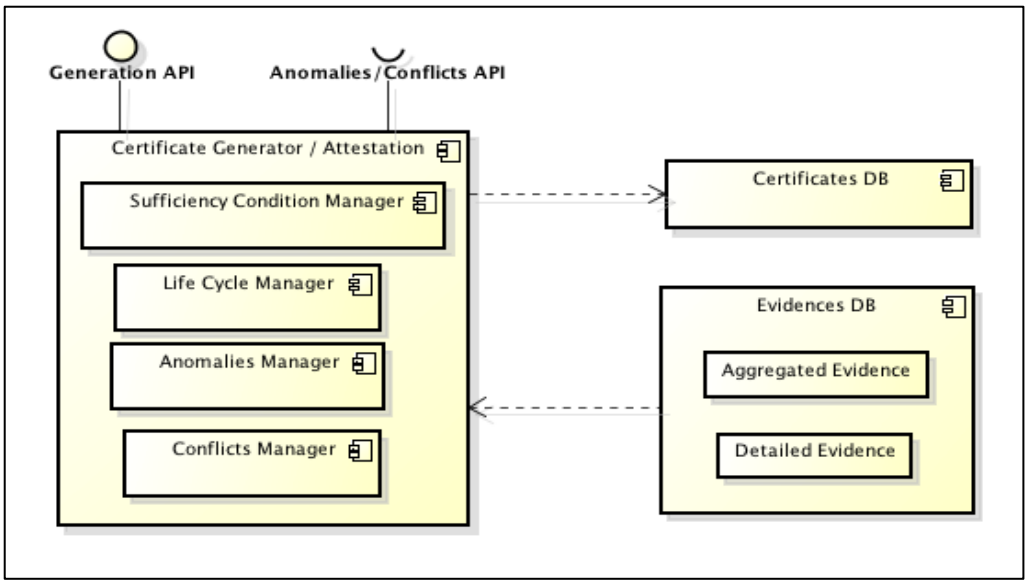


Figure 53 – Certificate Generator

Table 17 – Generation API

Generation API	
Operation	**Description**
`submitCertificationModel(String model)`	This method allows submitting a certification model to the Certificate Generator.

Parameters		
Name	**Type**	**Description**
model	String	String representation of an XML certification model.

Operation	**Description**
`changeStateOfCertificate(Long certificateID, String state)`	This method is responsible for changing the status of the certificate according to the "LifeCycle Manager".

Parameters		
Name	**Type**	**Description**
certificateID	Long	Long value of a certificate Id.
State	String	String value of the current state of the certificate

Operation	**Description**
`createcertificate(String Certificate_Id, String SecProperty, String TOC)`	This method is responsible for creating a certificate when conditions are satisfied.

Parameters		
Name	**Type**	**Description**
Certificate_Id	String	The unique identifier of the certificate
SecProperty	String	The security property that the generated certificate refers to.
TOC	String	The ToC that the generated certificate refers to.

According to the type of conditions, which should be taken under consideration in order to decide the state of the generated certificate, there are relevant sub-components of the CG that deal with them. Below we describe each sub-component and their functionality.

6.3.4.1 ANOMALIES MANAGER

This component is a sub-component of the CG component, and is responsible to detect if any anomaly stated in the Certification Model has occurred. As an anomaly, we define any potential attacks on TOC or any other suspicious behaviour related to the property, which despite not having caused any violation of the security property of the model so far, may lead to a violation of this property in the future.

Thus, in order to detect an anomaly, this component checks through the Detailed Evidence Database, where all the monitoring results sent by the monitor are stored, to find if there is evidence with an assertion Id that matches the anomalies assertion Id defined in the Certification Model.

Whilst an anomaly is being detected, the framework notifies the issuer, who is responsible to either resolve it or decide that it is a critical one and cannot be resolved, which will lead to the revocation of the certificate (if this is stated in the life cycle model). For the notification of the issuer, the CG will provide the detected anomalies to the Certification Communicator through Anomalies/Conflicts Manager API, which is the relevant component to communicate with the external actors, and will wait for the response about weather the anomalies were resolved or not.

6.3.4.2 CONFLICTS MANAGER

This component is responsible to detect if any conflict stated in the Certification Model has occurred. Similar to the Anomalies Manager component, it checks the detailed evidence in the database, and checks if there is any evidence with the conflict assertion type. If it detects a conflict it will notify the Certificate Communicator component through the Anomalies/Conflicts API and it will wait for the resolution response.

As a conflict we define any violation of the security property rule that might occur in a shorter period than the defined aggregation period. For example, in the "*Availability-percentage-of-up-time*" security property, it could be defined that a ToC should have an average up time of 99% in

a period of one month (*aggregation period*). However, if the availability is checked in a weekly basis, it might fell below 99%, which might not affect the overall percentage of the aggregation period. If a conflict occur the framework will notify the issuer and they will decide whether this conflict can be resolved or not. For the notification of the issuer, the CG component exposes the Anomalies/Conflicts Manager API to the Certification Communicator.

6.3.4.3 SUFFICIENCY CONDITION MANAGER

This component is responsible to check if enough evidence is collected, according to the Certification Model, in order to issue a certificate. This component can check the monitoring period defined in the CM or number of events that should be monitored before issuing a certificate. An additional check that can also be defined in this element of the CM is for the "*ExpectedSystemOperationModelCondition*" condition, which is used to define an expected operation model of ToC.

More specific, there might be cases where an expected behaviour of the TOC should be defined, that states which and how many events the monitor should check before issuing a certificate. To enable checks of the representativeness of monitoring events, the certification model includes a specification of a model for the expected behaviour of ToC. As we discussed in in Section *4.3.1.5.1.(C)*, in order to define the sufficiency conditions in the certification model, a state transition model might be defined to describe probabilities of occurrence of events that occur in the service (ToC). In order to handle this type of conditions we have developed an algorithm that first checks the total number of events received from the service, and then it checks if every type of events that are relevant for the monitoring process are within the limits of the expected frequency of each case, as defined in the model.

Below the algorithm that this component is using to check the *ExpectedSystemOperationModelCondition* is presented.

Table 18 - Algorithm for Processing Behavioural State Transition Models

```
1   CheckEvent(e, state, nstate, CountES[e,state], CountS[state], valid) {
2   // CountES[e,state] is the total num of occurrences of e in state
3   // CountS[state] is the total num of occurrences of any event in state
4     if there is t in state.transitions such that t.event = e then {
5       CountES[e,state] = CountES[e,state] + 1; CountS[state] =
6   CountS[state]
7       + 1; nstate = t.ds; valid = true
8     }
9     else
10      {valid = false}
11    }
12    Boolean UpdateCounts(trace){ //ValidPR[e,s] indicates the satisfaction
13     of expected frequency range of all events of all state transitions
14    Set CountES[e,s] to 0 for all states s and events e of its trans;
15    Set CountS[s] to 0 for all states s;
16    Set ValidPR[e,s] to false for all states s and events e of its trans;
17    CST = ETOCB.s0; //CST is the current state
18    NST = nil;
19    validTrace = true;
20    While not end of event trace and validTrace do {
21      e = next non processed event in trace;
22      CheckEvent(e, CST, NST, CountES[e,CST], CountS[CST], validTrace);
23      if validTrace {
24        for each t in CST.transitions do {
25          if (CountES[t.e,CST]/CountS[CST] in R(t.e.lpr, t.e.upr)) {
26            ValidPR[t.e,CST] = true
27          }
28        }
29       CST = NST
30     }
31    return (validTrace)
32  }
```

Symbols	
e – Event	CST – The current state
state – The current state	NST – The next state
nstate – The next state	t – Transition
CountES[e,state] – List of number of occurrences of e in state	t.e – Event of the transition t
CountS[state] - List of number of occurrences of any event in state	lpr – lower probability of ocurrance
ValidPR[e,s] – List of satisfaction of expected frequency range of all events of all state transitions	upr = upper probability of ocuurance
	R(t.e.lpr, t.e.upr) – the range of the expected relative frequency of undertaking this transition whilst in CST

Functions

CheckEvent(e, state, nstate, CountES[e,state], CountS[state], valid) - Is a function that checks all occurred events and counts the valid ones.

UpdateCounts(trace) – This function updates the relative frequency of each valid event in the current state, and updates the array ValidPR[e,s] that is used to indicate the valid frequencies of events in a specific state s.

As shown in Table 18, the algorithm first checks all events and if an event is valid, then the *UpdateCounts()* updates the relative frequency of it in the current state (see array *CountES[e,state]*). It also updates the array *ValidPR[e,s]* that is used to indicate if the expected frequency range of the current event *e* in state *s* is preserved by the current relative frequencies of events. Moreover, the *UpdateCounts()*checks if each next event in the event trace is consistent with the ordering of events defined in the ETOCB. If it is not, then the *UpdateCounts()* reports the trace as invalid. For example, if an event is relevant for the specific security property defined, the event is being counted in the ETOCB, whereas in cases where an event is not relevant for the specific checking (i.e. different kind of event occurred in the TOC), but it was received from the monitored service, then the *UpdateCoutns()* function will report it as invalid, and would not be counted for the ETOCB.

Thus, if an expected behavioural model is defined, in order to check this type of conditions, this component executes the algorithm presented above. According to this algorithm, the sufficiency Condition Manager component checks if all possible paths are covered before issuing a certificate. To do so, it checks the *PrimitiveEvents* table of the database, where all events occurred in the ToC are being stored. Then by checking the different types of these events (calls or responses), as well as the sender and the receiver of each event, it can keep track of every event that took place and compare it with the conditions defined in the behavioural state transition model.

6.3.4.4 LIFE CYCLE MANAGER

This component is responsible to handle the Life Cycle Model defined in the monitoring based certification model. In order to generate a certificate, this sub-component of the CG checks the conditions that apply to the particular transaction in the certificates life cycle of the certification model, which leads the certificate to a specific state. If the required conditions are fulfilled, such as if there are enough monitoring results produced or if the specified monitoring period has passed, then the CG will set the certificate's state to the relevant one. To do so, it pulls regularly data from the Evidence DB, and checks whether the conditions are met.

Since the certificates life cycle is defined with states and transactions that lead from a specific state to a different state, each transaction refers to a specified condition (with the reference Id of each condition). Therefore, the CG will check every condition specified in the *Assessment Scheme* of the Certification Model and will combine it with the relevant transaction as specified in the life cycle, so as every time a condition occurs, is will update the state of the generated certificate.

The framework uses the Life Cycle Model (LCM) of a certification model in order to monitor the overall certification process and to update the status of certificates, which may be generated according to it. More specifically, starting from the initial state of the LCM the framework will process all events according to this model. This processing is based on the algorithm provided below.

Table 19 – Algorithm for Processing the Life Cycle Model

```
1   State ChooseTransition(State curstate, EventQueue queue){
2     top = queue.head(); //returns null when queue is empty
3     tr_ev = {t ∈ transitions(curstate): top≠null && t.event()=top};
4     //trans matching events
5     tr_em = {t ∈ transitions(curstate): t.event() = ""}; //trans with no
6                                                          events
7     en_ev = {t ∈ tr_ev: satisfied(guard(t))};//trans with True guard & match
                                               event
8     en_em = {t ∈ tr_em: satisfied(guard(t))};//trans with True guard but no
                                               event
9     t = null;
10    if(en_ev =0&&en_em =0){
11      if(top≠null) throw invalidEvent; //non matching event from the queue
12    else return(curstate);
13    } else if (en_ev≠0) {  //select transition with event
```

```
14      t = select (en_ev); queue.pop();
15    } else { //select transition with True guard but no event
16      t = select (en_em);
17    }
18    for (a : retrieveActions(t)) { //retrieve transition actions
19       execute(a); //execute actions
20    }
21    return t.nextState(); //return the new state
22  }
```

Symbols	tr_{em} – transition with no events
curstate – The current state	en_{ev} – transition with matching event and satisfied guard condition
queue – queue with events	en_{em} – transition with no event and satisfied guard condition
top = queue.head()	t – Transition
tr_{ev} – transition with events that matches events of the queue	a - Action

Functions

ChooseTransition(State curstate, EventQueue queue) – This function checks if there is an event in the queue that matches an event of a transition of the current state and then for the specific transition checks if the guard condition (if any) is satisfied, in order to move the next state that this transition leads to.

According to the algorithm presented in Table 19, all the events received by the TOC during the certification process are placed in a queue. An event can be a condition that is met (e.g., *EvidenceSufficiencyCondition*, *aggregation period*, *expiration condition* etc.). The algorithm checks if there is an event in the queue that matches an event of a listed transition of the current state of the LCM and if the guard condition of it (if any) is satisfied. When these conditions are satisfied for the specific transition, the algorithm executes the actions for the transition, and sets the status of the certificate that is being handled by the process, to the state that the transition leads to. To check the conditions associated with the transitions of an LCM, the algorithm pulls regularly data from the database storing the monitoring evidence gathered, and checks the conditions against it.

6.3.5 MONITORING MODULE

The monitoring module presented in Figure 54 is responsible for monitoring a specific security property, as defined in the Certification Model, against runtime events of a system. This module receives the EC-Assertions for a specific ToC and Security Property by the Monitoring Manager component, and more specifically form the Monitor Translator, in order to start monitoring the events of the ToC. The events are being captured by the Event Captor component and received through the Event Bus component. Moreover, when the monitoring process starts, the monitor will start producing monitoring results that should be send to the framework for the certification process. This is done through the Monitoring API. The method implemented at the Monitoring API is described in Table 20.

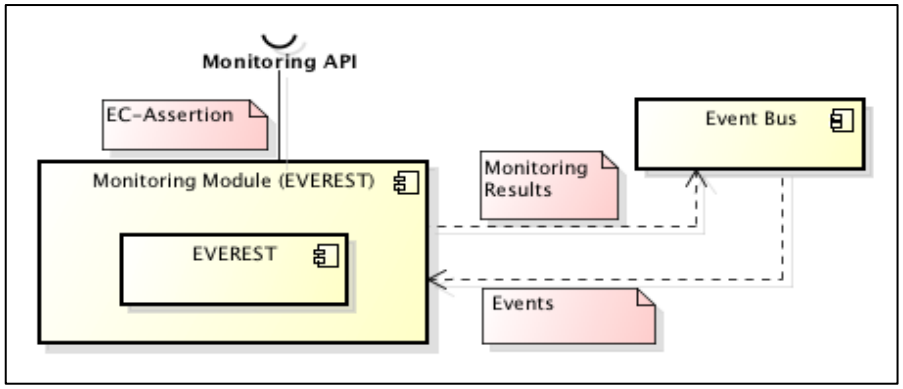


Figure 54 – Monitoring Module

Table 20 – Monitoring Module API

Monitoring API	
Operation	**Description**
`List<String> getLatestMonitoringResults()`	This method allows retrieving the latest monitoring results from the monitoring module.
Parameters	

Name	Type	Description
Return Type	List of String	String representation of the XML monitoring result.

6.3.6 EVENT CAPTOR

The Event Captor works as a channel between a Web Service Client, a Web Service Container and an Event Receiver (Event Bus component), as it is shown in the components diagram in Figure 55. The Event Captor that we have used is mainly for Web services, but our certification framework could work with any type of event captors in a cloud environment, as long as the format of events generated by for monitoring complies with format that the monitor understands. Thus, the current Event Captor listens to any SOAP request that arrives at the port and forwards them to the Web Service container (tunnelhost, tunnelport). It also creates and sends the captured events to the Event Bus (receiverhost, receiverport).

Any SOAP response received from web service container is forwarded to the web service client (listener port) and the generated events are forwarded to Events Receiver (*receiverhost*, *receiverport*).

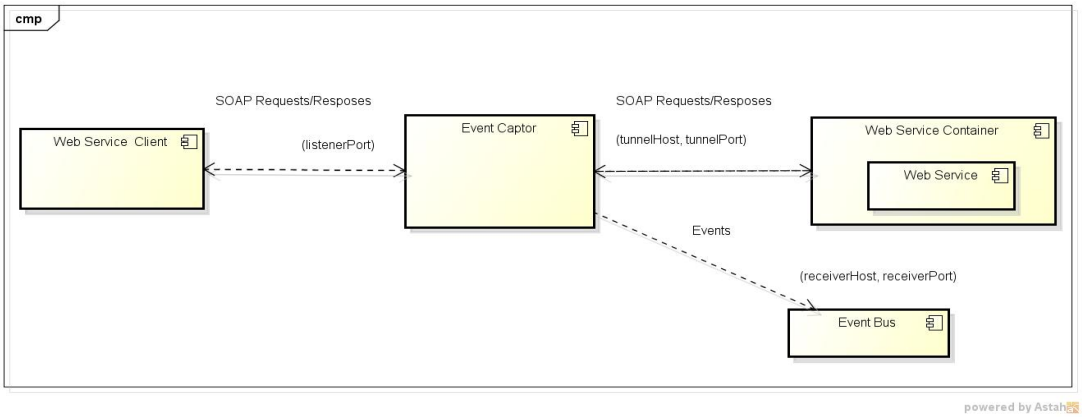


Figure 55 – Event Captor

6.3.7 EVENT BUS

The Event Bus component is a "publish/subscribe" event communication infrastructure that is used to forward any event captured by the Event Captor, to the framework. More specific, after locating a suitable monitor in a specific cloud, the framework gets from it a token assigned to an event channel of interest and uses this token to subscribe the monitor to the Event Bus. The same token is passed to the Event Captor to be used when it publishes events to the bus so that these events can be forwarded from the appropriate monitor.

The Event Bus component support several technologies/protocols related to messaging implementation, namely: XMPP, JMS [45] and AMQP [107], but for our purposes only XMPP will be used.

6.3.8 EVIDENCE DATABASE

The Evidence Database is used to record all types of evidence (primitive events or monitoring results) and all their details, in case they will be needed for auditing purposes. All tables are built based on MySQL.

6.3.8.1 DETAILED EVIDENCE

This table presented in Table 21 records the monitoring results received from the Monitor that is collected according to Certification Model, regarding the assessment of security properties.

Table 21 – Detailed Evidence Table

Columns	Description
Event_Id	A unique identifier (Id) for every event captured
CM_Instance_Id	Reference to the Certification Model instance that the aggregated evidence relates to (Foreign key referencing Certification Model DB)

TimeStamp	The time of the evidence collection
EventPayloadType	Type of evidence: • InteractionEventType (the evidence that captures a call of a service operation or a response from the execution of a service operation) • MonitoringResultEventType (a monitoring result regarding the satisfaction or not of a security property that is produced by a monitor) • PredictionResultEventType (a prediction regarding the satisfaction or not of a security property by the end of a specific period of time in the future that is produced by the monitor) • InfrastructureMonitoringEventType (a measure regarding some infrastructure layer property)
Evidence_ XML	The XML encoding of the evidence, represented as a string.
Evidence_ Object	The evidence as java object, which is automatically produced by the monitoring manager before getting recorded in the database.

A. Sender

This table presented in Table 22 records the sender of each primitive event that is collected, regarding the assessment of security properties.

Table 22 – Sender Table

Columns	Description
Event_Id	Reference to a specific event captured (Foreign key referencing the Detailed Evidence table)
Name	The name (Id) of the sender
IP	The IP address of the sender
Port	The port number used by the sender process
User_Id	A unique identifier (Id) of the user that started the sender process
ProcessId	A unique identifier (Id) of the process that generates the event

B. Receiver

In Table 23 is presented the Receiver table, which records the receiver of each primitive event that is collected, regarding the assessment of security properties.

Table 23 – Receiver Table

Columns	Description
Event_Id	Reference to a specific event captured (Foreign key referencing the Detailed Evidence table)
Name	The name (Id) of the receiver
IP	The IP address of the receiver
Port	The port number used by the receiver process
User_Id	A unique identifier (Id) of the user that started the receiver process
ProcessId	A unique identifier (Id) of the process that generates the event

C. Notifier

This table presented in Table 24 records the notifier (event captor) of each primitive event that is collected regarding the assessment of security properties.

Table 24 – Notifier Table

Columns	Description
Event_Id	Reference to a specific event captured (Foreign key referencing the Detailed Evidence table)
Name	The name (Id) of the notifier
IP	The IP address of the notifier
port	The port number used by the notifier process

An example of a monitoring result in XML is given below, in order to provide the way payload is defined of the detailed evidence. It is showed that the Event Type is *MonitoringResultEventType*, and that the result is a violation. The event metadata signifies that 100 events were used to derive this monitoring result.

```
<eventInstance xmlns="http://www.slaatsoi.org/eventschema">
    … … … …

  <EventPayload>
    <MonitoringResultEvent>
      <SecurityPropertyInfo assessmentResult="violation"
                cmInstanceID="1001-inst" securityPropertyID="1001-inst">
        <GuaranteedState assessmentResult="violation"
           guaranteedID="ORCThroughputConstraintInventoryBookSaleState">
          <QoSName>http://www.slaatsoi.org/commonTerms#arrival_rate
          </QoSName>
          <QoSValue>0.008264462809917356</QoSValue>
        </GuaranteedState>
      </SecurityPropertyInfo>
    </MonitoringResultEvent>
  </EventPayload>
  <EventMetadata>
    <key>NumberOfEvents</key>
    <value>100</value>
  </EventMetadata>
</eventInstance>
```

6.3.8.2 PRIMITIVE EVENTS

In this table the events received by the Event Captor from the ToC, regarding a specific CM, are being stored. Table 25 shows the structure of this table.

Table 25 - Primitive Events Table

Columns	Description
EventId	A unique identifier (Id) of the primitive event
TimeStamp	The time of the evidence collection
ecName	The name of the event based on the EC-Assertion type, such as Happens.

Prefix	The prefix of the EC-Assertion event, such as *ic* for call or *ir* for a reply event.
Operation name	The name of the operation of the event
PartnerId	The name of the service interface of the event.
Sender	The sender of the event.
Receiver	The receiver of the event.
negated	Whether the event is negated
Abducible	Defines if abductive reasoning generated the event.
Recordable	Defined if the event was recorder to process the predicate defined in the EC-Assertion
eventObject	The event as java object, which is automatically produced by the monitoring manager before getting recorded in the database.
eventString	The XML encoding of the evidence, represented as a string.

6.3.8.3 AGGREGATED EVIDENCE

This table that is presented in Table 26 records the aggregated evidence, which is generated by the "Aggregation Manager" component according to the CM and it is inserted in the generated certificates.

Table 26 – Aggregated Evidence Table

Columns	Description
AggregatedEvents_Id	A unique identifier (Id) of the aggregated evidence
CM_Instance_Id	A reference to the Certification Model instance that the aggregated evidence relates to (Foreign key referencing Certification Model DB)
Creation_Time	The creation time of the aggregated evidence
Start_Time	The start time of the aggregation period

End_Time	The end time of the aggregation period
Assertion_ID	A reference to the Assertion ID that expresses the security property for which this aggregation is produced (Foreign key referencing Certification Model Table)
Assessment_Result	The overall assessment result of the security property, which is either *satisfied* or *violated*.
Evidence_ XML	The aggregated evidence as string representation of XML
Evidence_ Object	The aggregated evidence as java object

Below, an example of an XML representation of the aggregated evidence is presented. In the *reportInfo* element is defined (i) the creator, (ii) the start date of the aggregation, (iii) the end date of the aggregation, and (iv) the Certification Model Instance that was used to assess the property ("*reportId*="Report-1001-inst"). Moreover, the *AssessmentResultSummary* signifies that for the *Guaranteed* of the monitored property, 58 violations were detected. Finally, the *FunctionalAggregatorResult* element shows the measure used to aggregate the monitoring results of the security property defined in the "1001-inst" certification model instance, which was the "average" value.

```
<aggregatedReportType xmlns:ns2=http://www.slaatsoi.org/eventschema
                      xmlns="http://www.slaatsoi.org/business-report-schema">
  <ReportInfo reportCreatorId="SLA@SOI Business Reporting"
              endTime="2013-08-29T15:32:57.136+01:00"
              startTime="2013-08-29T15:32:57.073+01:00"
              timestamp="2013-08-29T15:32:57.136+01:00"
              reportId="Report-1001-inst"/>
  <AssessmentResultSummary>
    <SecurityProperty notAssessted="0" satisfactions="0" violations="58"/>
    <Guaranteed notAssessted="0" satisfactions="0" violations="58"/>
  </AssessmentResultSummary>
  <FunctionalAggregatorResultSummary>
    <FunctionalAggregatorResult aggregateValue="0.0018021085940434745"
          functionalAggregatorId="Average"
          guaranteedTermId="ORCThroughputConstraintInventoryBookSaleState"
          securityPropertyId="1001-inst"/>
  </FunctionalAggregatorResultSummary>
</aggregatedReportType>
```

6.3.9 CERTIFICATION MODEL DATABASE

Table 27 present the certification model database, in which all certification models that are used to assess security properties are recorded. For every request of a specific certification model (*CM_Id*), a new CM Instance (*CM_Instance_Id*) will be created.

Table 27 – Certification Model DB

Columns	Description
CM_Id	A unique identifier (Id) for every CM - primary key
CM_Instance_Id	A unique identifier for every request made by a CA to apply a specific CM to a specific ToC.
CASignature	The CA signature of a specific instance of the CM
Security_Property	The security property category that is going to be certified
Assertion_ID	A unique identifier (Id) of the assertion that expresses the property
Assertion	The specification of the security property in the certification language
ToC_ID	The unique identifier (Id) of the Target of Certification (ToC) that is being certified by the specific instance.
ToC_Name	The name of the ToC that is being certified by the specific instance
CM_ XML	The whole CM as string representation of XML
CM _Object	The whole CM as java object

6.3.10 CERTIFICATES DATABASE

In Table 28 the Certificates Database table presented, which records all the generated certificates, in order to be provided in the requestors or notify the registered users for every update that might occur.

Table 28 – Certificates DB

Columns	Description
certPK	A unique identifier (Id) for every certificate generated (cert) - primary key
certSerialNo	The serial number of the generated certificate
Security_Property	The security property category that is being certified
tocName	The name of the ToC that is being certified by the specific instance (Foreign key referencing Certification Model Table)
validFrom	The date from which the certificate begins to be valid
validUntil	The expiration date of the certificate
certString	The whole certificate (cert) as a string representation of XML
certObject	The whole certificate (cert) as java object

Chapter Seven

EVALUATION

7.1 OVERVIEW

In this chapter we present the process that we have followed in order to evaluate the certification framework introduced in this thesis and the outcomes of this evaluation. The evaluation has been based on the prototype tool that we developed to implement the framework (see Chapter 6), the EVEREST monitoring tool used in the prototype, and for the verification of the proposed certification model we have used the Prism model checker [191].

7.2 EVALUATION METHODOLOGY

In order to evaluate all the aspects of this research, three different activities were chosen, as there are three major contributions. Thus, we selected to evaluate first the proposed Certification Model that was introduced earlier in Chapter 4, and is used as an input in the certification process. Then, the defined Life Cycle model in the CM was also evaluated, to check its correctness for handling the process and updating the status of the certificates. Finally, the whole framework was also evaluated in terms of correctness and performance at run time.

The three separate activities that our evaluation is based on, focusing on different aspects of our approach. These activities were:

1. ***Subjective evaluation of the comprehensiveness and complexity of certification models for certifiers, and whether they cover all the need for a certification process of a cloud service.***

This activity was based on interactive sessions with experts in certification and the use of a questionnaire that they had to fill in order to give their feedback about the complexity and comprehensiveness of the certification models used in our approach.

2. ***Formal analysis and verification of certification models.***

   This activity focused on the use of model checking in order to verify properties of the certification processes (i.e., life cycle model) that are specified in the proposed certification model. This analysis was based on a symbolic analysis and a model checking using the PRISM model checker.

3. ***Experimental evaluation for the operational correctness and performance of the certification framework at run time.***

   This activity focused on the investigation of the performance and correctness of the certification framework, at runtime. The activity was based on the use of a certification model for database management systems (DBMS) developed based on a Protection Profile of Common Criteria, and the use of a DBMS as part of an e-commerce system, established by a benchmark used for performance evaluation in this area.

7.3 EVALUATION ACTIVITIES

Different sets of experiments and evaluation activities were conducted in order to evaluate our research. Below each type of activity that was conducted is being presented and explained, as well as the analysis of their results.

7.3.1 SUBJECTIVE EVALUATION

For this type of evaluation, we conducted a survey to four certifiers, after having presented them the proposed monitoring-based certification process and explained them the proposed certification model.

7.3.1.1 SUBJECTIVE EVALUATION METHODOLOGY

The four persons that were chosen to answer to our questionnaire are Italian certifiers. The session was part of the evaluation of the CUMULUS project, thus the certifiers were chosen by a project partner based on the availability of relevant experts to take part to our survey.

The session started with explaining the session structure and objectives to the participants and followed by an introduction of the relevant aspects of the certification process presented. Then, according to the planned duration of the overall session, each section was executed separately, lasting about 15 to 30 minutes for a slide presentation of the certification process and of the relevant certification model. Regarding our monitoring-based certification process, the presentation included an introduction to both structure and objectives of the monitoring based certification model, with a main focus on how it attempts to capture the basic concepts of the monitoring process, as well as the definition of the different kind of conditions that need to be defined in the certification model and checked by the proposed framework.

After the presentation, the participants had 15 minutes to answer the questionnaire. Moreover, further details for clarification purposes were given when asked by the participants and where needed, they were requested to clarify their answers to the questionnaire, by providing written comments.

The questions were only closed-answer questions of two possible types: i) questions with a Yes/No answer with a request to provide a written explanation, and ii) questions with five level scale answer (Excellent, Good, Neutral, Poor, Very Poor). We have also provided a free text space for each question for any further comment they would like to add, so as to gather as much feedback as possible from the certifiers. Moreover, we decided to provide a free text space also for the overall session, to allow the participants to report any extra relevant comment with regards to the overall process or provide comments that were not included to the questions.

It is important to notice that we had a time constraint due to the actual availability of the certifiers, thus we needed to restrict our session to present only in a conceptual level the certification process and to arrange only one session that would last no more than half a day. The main effect on the session design were:

- Presenting the framework based on only one use case and corresponding functional and security requirements, and
- Presenting a simplified version of the Monitoring Based Certification Model.

These lead to a partial satisfaction of the general evaluation criteria, as well as to the fact that there was no time to for the participants to attempt to produce a certification model or to have a more detailed training on how to define it. Thus, not all certifiers understood completely the monitoring process, or they found some elements of the certification model too complicated to define.

The questions were formed in order to cover i) the user satisfaction aspect with regards to the concept of *easiness of comprehension* of the certification model (Question 1), and ii) the representation capability analysis by exploring the ability of the Monitoring Based Certification Model to capture the significant aspects of the monitoring based certification process (Question 2-5) and by gathering the overall rating for the ability of the certification model to capture significant aspects of security certification of cloud services (Question 6).

Moreover, there were some comments about changes that might occur in a service that is being monitored and certified and whether we can detect these changes in order to adapt the certification process according to them. Finally, some certifiers also proposed to combine the monitoring-based certification process with a test-based process, to check that no changes have occurred in the service being certified.

7.3.1.2 ANSWERS OF THE QUESTIONNAIRE

A more detail presentation of the answers in each question of our questionnaire is provided and analysed below. Table 29 below shows all answers of each participant to the different questions.

Table 29 - Answers of the Questionnaire

	Q1	Q2	Q3	Q4	Q5	Q6
P1	>90%	N/A	>90%	N/A	Assertions	Yes
P2	>90%	>90%	>90%	>90%	None	No
P3	>90%	>90%	>90%	>90%	Assertions	Yes
P4	>90%	>90%	>90%	50-74%	Evidence Sufficiency Conditions	No

Below all questions are presented and a further analysis of the given answers is provided.

Question 1

Do you think that Monitoring Based Certification Models (MBCMs) are capable of representing comprehensively continuous security certification processes for cloud services security?

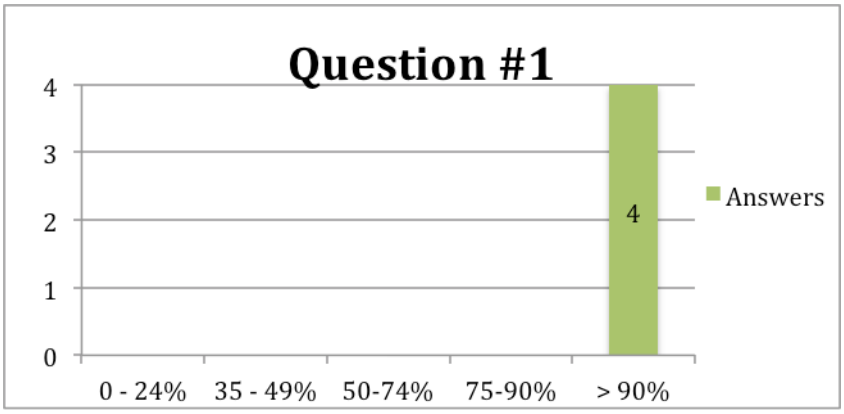


Figure 56 – Answers to Question 1

The first question of the questionnaire was about the certification model's capability to offer continuous certification for cloud services. As shown in Figure 56, all four certifiers replied that the MBCM is able to provide continuous certification for cloud services.

Question 2

Do you think that the assertion rules specified as part of a the Monitoring Based Certification Model are capable of representing accurately and effectively the continuous collection of evidence required for the assessment of security properties and/or the effectiveness of control mechanisms realising these properties in the cloud?

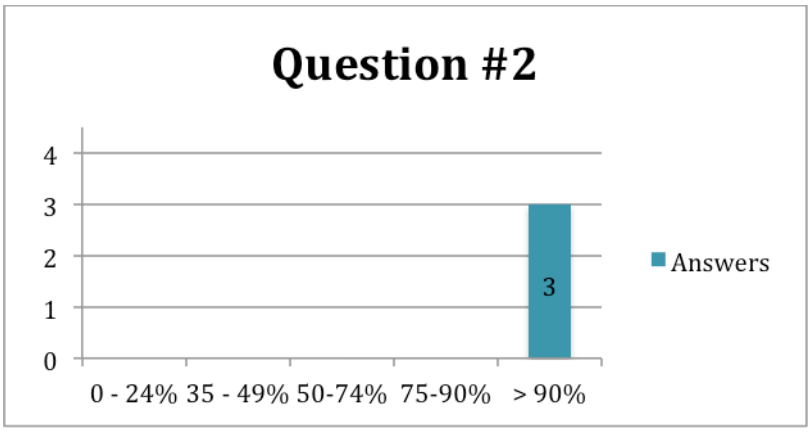


Figure 57 – Answers to Question 2

In this question all certifiers except one, as shown in Figure 57, replied. All three of them answered that the monitoring assertion rules defined in monitoring based certification models are capable to accurately specify the continuous monitoring process for assessing most security properties. The only certifier who could not answer this question indicated that this was due to lack of knowledge and sufficient information about the monitoring process, thus was unable to make a statement for the assertion language.

Question 3

Do you think that the life cycle models specified as part of a Monitoring Based Certification Model are capable of representing effectively the processes of collecting evidence, and generating and managing certificates based on it?

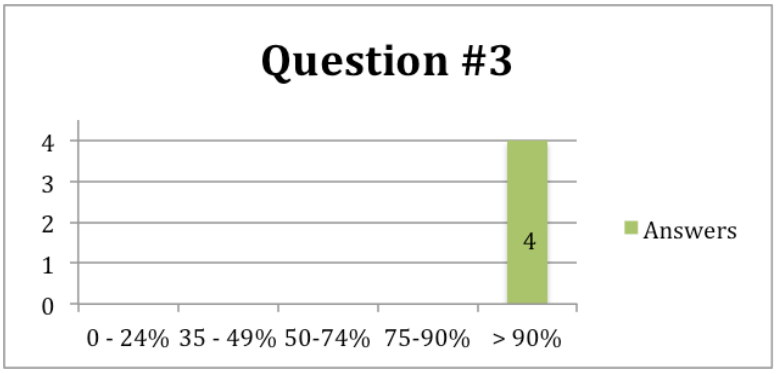


Figure 58 – Answers to Question 3

All certifiers stated that the life cycle model of the certification model is able to effectively represent the certification process, as shown in Figure 58. However, there was also some uncertainty in their answer because two of them found the process complicated and difficult to understand all steps.

Question 4

Do you think that the evidence sufficiency conditions that may be specified as part of a Monitoring Based Certification Model (number of events, period of monitoring, expected behaviour of target of certification) are capable of representing effectively the circumstances under which the evidence collected would be enough to make a decision about issuing a certificate or otherwise?

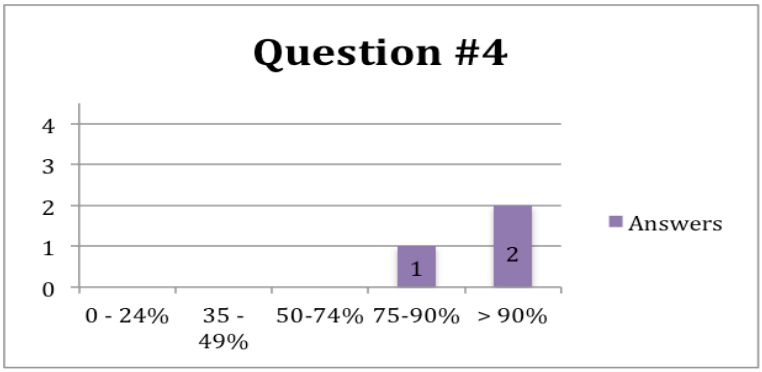


Figure 59 – Answers to Question 4

As shown in the Figure 59, three out of four answers in this question stated that the element used to specify the sufficiency conditions in the certification model can be used effectively to specify the circumstances, under which the collected evidence are enough in order to issue a certificate. However, one certifier answered that we should take under consideration the cases where changes might occur in the service during the monitoring process, thus this element is not sufficient for these cases.

Question 5

Which of the following parts of the Monitoring Based Certification Models (Assertions, Evidence Sufficiency Conditions, Life-Cycle Model) do you think that it would be difficult for someone with expertise in cloud security to specify even after training?

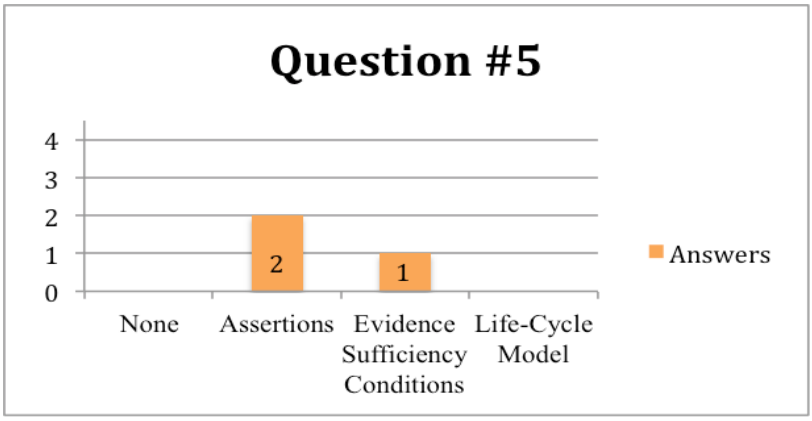


Figure 60 – Answers to Question 5

As Figure 60 presents, half of the certifiers found difficult to specify the Assertion element, which is used to express the monitoring rules to collect evidence for security properties and anomalies. Another answer was that the most difficult element to define is the evidence sufficiency conditions. One certifier was unable to answer, stating that it was difficult to understand the whole process.

Question 6

Are there any key elements/requirements that continuous security certification processes for cloud services should address but the Monitoring Based Certification Models fail to cover?

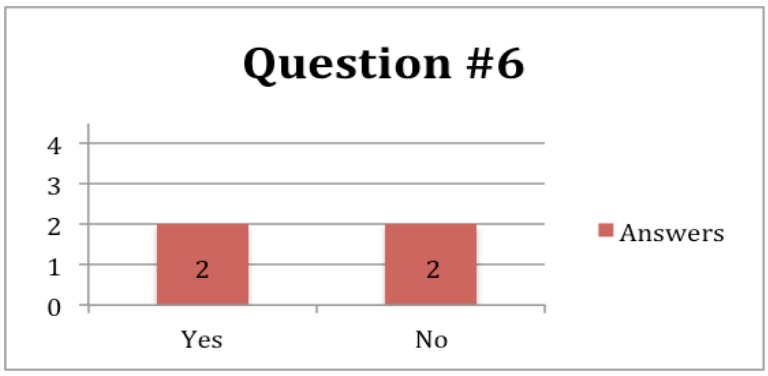


Figure 61 – Answers to Question 6

In this question there were comments concerning the adaptation of the framework to possible changes in services that are being certified, and whether the monitor can detect them in order to continue the certification process according to them. Another comment was about including some functional tests, to assure that no changes have been made at the services.

During the session, the participants asked questions regarding the monitoring process and the specification of some elements. More specifically, according also to the answers in Question 5, they found more difficult to specify the Security Property Assertion element and the Evidence Sufficiency Conditions elements. This is due to the lack of time to explain in more details the EC-Assertion language and its elements, as well as the expected behavioural model, which they found difficult to understand the concept of it. Furthermore, during the presentation of the monitoring-based certification process, questions arose regarding the expected behavioural model and the life cycle model and the way the framework process the state machines defined in the CM for these two elements. Moreover, also during the time of answering the questionnaire, some participants asked for further clarification of some questions, as well as to re-explain some parts of the presentation, to refresh their memories to answer the questions.

Overall, the participants seemed to understand the process, after further clarifications, and found it really interesting and useful for certifying cloud services. However, one participant was

confused with the whole process and had difficulties in understanding the elements to be defined in the CM and in particular the way to specify them.

7.3.1.3 THREATS TO VALIDITY

For this subjective evaluation concerning the comprehensiveness and correctness of the proposed certification framework, different factors affected the results. Firstly, one main limitation in this activity was the lack of availability of relevant expert personnel to participate in our survey. Thus, we could only involve four Italian certifiers to participate and answer our questionnaire.

Furthermore, another limitation was the time constraint of the participants. Since we have limited time to present and explain the monitoring based certification process, as well as the certification model that we have defined, we were able to present it in a conceptual level, without going into more technical details. Thus, there was no possibility to train the participants or engage them to try and define a certification model on their own.

Based on these limitations, we were able to provide a presentation with as many details as possible, on the certification process and on the certification model, by referring to only one case study as an example.

As a result, the provided explanation was not adequate leading to the fact that one participant was not able to fully understand the process and was unable to answer to some questions. Thus, not all answers of this survey were indicative of our work.

7.3.2 VERIFICATION AND FORMAL ANALYSIS

For conducting the verification and a formal analysis of the proposed life-cycle model of the certification model, we used the Prism model checker tool used for formal modelling and analysis of systems that perform random or probabilistic behaviour [191].

7.3.2.1 FORMAL ANALYSIS METHODOLOGY

In order to check the correctness of the defined Life-Cycle model of the Certification model, we used the Prism model checker. In Prism, we define three different modules to check the correctness of the model. These modules are: the: i) Monitoring Manager, ii) Anomaly Manager, and iii) Monitor.

As shown in the figure below, different states are defined to denote whether a certificate is issued (*State 2*) or not (*State 1*) in the Monitoring Manager module, and in every state the module receive different types of events from either the monitor or the anomaly manager module. Based on the type of events and the conditions defined in the Life Cycle model of the CM, a certificate could be issued (cert=1), revoked (cert=2) or not issued (cert=0).

The check starts from the Monitoring Manager module, which as a first step initiates the Monitor to start sending events (*start=1*). As soon as the Monitor is initialised and moves from *State=0* to *State=1*, there are different paths that it can take with different probability for each one of them. For this example presented below, we have chosen the following probabilities, for testing purposes to check the life cycle model: i) a probability of 90% to send a rule satisfaction and 10% to send a rule violation when it process the security property assertion, and ii) a probability of 90% not to detect an anomaly (thus, the assertion of the anomaly is not satisfied) and 10% to send an anomaly event, whilst processing the anomaly assertions.

In case of a rule satisfaction (*ev=1*), the Monitoring Manager keeps record of this type of events by increasing the variable *nsat* every time it receives a rule satisfaction. Similarly, in case it receives a rule violation (*ev=2*) it increases the variable *nvio* and in case that no anomaly is being detected (*ev=6*) it increases the variable *noAnom*. If an anomaly is received (*ev=3*), then except of increasing the variable *nan*, it also starts the Anomaly Manager module in order to see if the detected anomaly will be resolved or not (*start=2*). When the Anomaly Manager is being initialised, then there is a probability of 80% to resolve the anomaly and 20% not to resolve it.

Thus, when the Monitoring Manager receives an event from the Anomaly Manager for a resolved anomaly (*ev=4*), the *resAnom* variable is increased, whereas in the case of an unresolved anomaly (*ev=5*), the variable *Unres* will be increased and the certificate will be either revoked

(*cert=2*), in case it was already issued when the unresolved anomaly event was received, and the monitoring process will terminate (state=3), or a certificate will not be issued (*cert=0*) and the monitoring process will terminate (state=3).

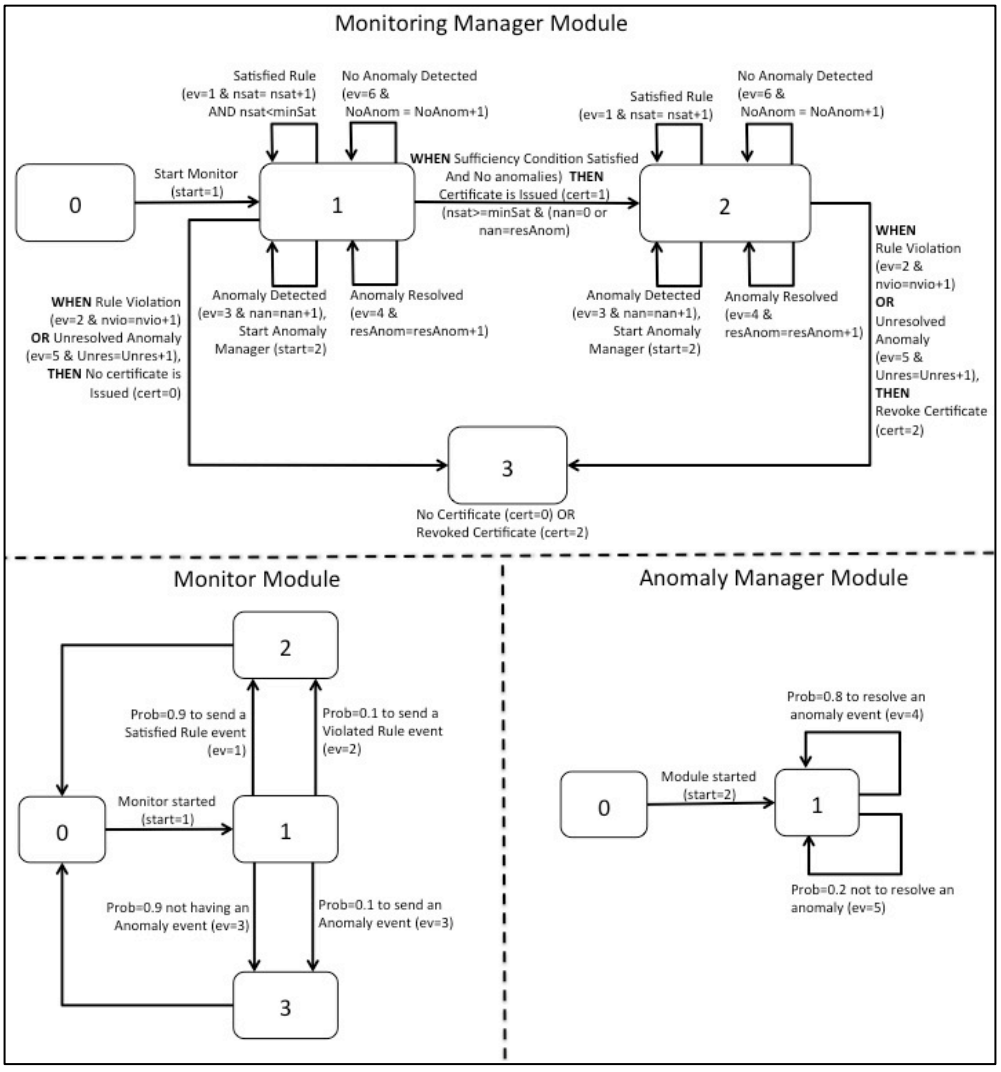


Figure 62 - Prism Modules

Below we provide the way these modules are defined in the Prism model checker.

```
dtmc
global start: [0..2];
global minsat: [0..60] init 10; //sufficiency condition: min num of
                                //satisfactory results
global resAnom: [0..60] init 0; //resolved anomalies
global ev: [0..6] init 0;     //monitoring evidence: 1(satisfaction), 2(violation),
                              // 3(anomaly), 4(resolved anomaly),
                              // 5 (unresolved anomaly), 6 (no anomaly)
global nan: [0..60] init 0;  //number of anomalies
global cert: [0..2];         // 0 (not issued), 1 (issued), 2 (revoked)


module monitoring_manager

   sr: [0..3] init 0; //states of the monitoring manager
   noAnom: [0..60] init 0;    //number of no anomalies detected
   nsat: [0..60] init 0;      // number of rule satisfactions
   nvio: [0..10] init 0;      //number of rule violations
   Unres: [0..10] init 0;     //number of unresolved anomalies

   [] sr=0 -> (sr'=1) & (start'=1);   //the monitoring manager initiates the monitor

   [] sr=1 & (ev=1) -> (nsat'=mod(nsat,60)+1) & (ev'=0) & (start'=1) & (sr'=1);
     //when a rule satisfaction is received nsat is increased by 1
   [] sr=1 & (ev=2) -> (nvio'= mod(nvio,10)+1) & (cert'=0) & (start'=0) & (sr'=3);
     //when a violation is received, nvio is increased by 1 and no certificate is
     //issued, the process stops (move to state sr=3)

   [] sr=1 & (nsat>=minsat) & (nan=0) -> (cert'=1) & (start'=1) & (sr'=2);
     //when sufficient evidence is reached and there are no anomalies,
     //a certificate is issued (cert=1) and moves to state sr=2

   [] sr=1 & (nsat>=minsat) & (nan=resAnom) -> (cert'=1) & (start'=1) &
      (sr'=2);   //when sufficient evidence is reached and there are no unresolved
                 //anomalies, a certificate is issued (cert=1)

   [] sr=1 & (ev=3) -> (nan' = mod(nan,50) + 1) & (ev'=0) & (start'=2) & (sr'=1);
      //when an anomaly is received, the anomaly manager is initiated and
      //the nan is increased by 1

   [] sr=1 & (ev=6) -> (noAnom' = mod(noAnom,50) +1) & (ev'=0) & (start'=1) &
   (sr'=1);  //when the anomaly rule is not satisfied the noAnom is increased by 1

   [] sr=1 & (ev=4) -> (resAnom' = mod(resAnom,50) + 1) & (ev'=0) & (start'=1) &
   (sr'=1);  // when an anomaly is resolved the resAnom is increased by 1

   [] sr=1 & (ev = 5) -> (Unres' = mod(Unres,10)+1) & (cert'=0)  & (start'=0) &
   (sr'=3);    //when an unresolved anomaly is received, an no certificate
               //has been issued, the process stops

   [] sr=2 & (ev=1) -> (nsat' = mod(nsat,50)+1) & (ev'=0) & (start'=1) & (sr'=2);
   //when a certificate is issued and rule satisfaction is received,
   //the nsat is increased by 1
```

```
   [] sr=2 & (ev=2) -> (nvio' = mod(nvio,10)+1) & (cert'=2) & (start'=0) & (sr'=3);
   //when a certificate is issued and a violation is received,
   //the certificate is being revoked (cert'=2)

   [] sr=2 & (ev=4) -> (resAnom' = mod(resAnom,50) + 1) & (ev'=0) & (start'=1) &
   (sr'=2);    //when a certificate is issued and a resolved anomaly is received,
               //the resAnom is increased by 1

   [] sr=2 & (ev=5) -> (Unres' = mod(Unres,10)+1) & (cert'=2) & (start'=0) &
   (sr'=3);    // when a certificate is issued and an unresolved anomaly event
               //is received, the certificate is being revoked (cert=2)
   [] sr=2 & (ev=3) -> (nan' = mod(nan,50)+1) & (ev'=0) & (start'=2) & (sr'=2);
     //when a certificate is issued and an anomaly is being detected,
     //the nan is increased by 1

   [] sr=2 & (ev=6) -> (noAnom' = mod(noAnom,50)+1) & (ev'=0) & (start'=1) &
   (sr'=2);  //when a certificate is issued and no anomaly is being detected,
             //the noAnom is increased by 1
   [] sr=3 -> (start'=0);
endmodule



module anomaly_manager
// the anomaly manager sends events concerning the anomalies resolution

   sp: [0..1] init 0;

   [] sp=0 & (start=2) -> (sp'=1);   // anomaly_manager is initiated by the
                                     //monitoring manager
   [] sp=1 & (nan>resAnom) -> 0.8: (ev'=4) & (sp'=1) // prob of anomaly resolution
                            + 0.2: (ev'=5) & (sp'=1);// prob for unresolved anomaly
endmodule



module monitor
       // the monitor sends security satisfaction, violation and anomalies event

   sm: [0..3] init 0;   //states of monitor
   counter: [0..50] init 0;  //number of sent events

   [] sm=0 & (start =1) -> (sm'=1);
     //monitor initiated by the monitoring manager
   [] sm=1 -> 0.90: (counter'=mod(counter,50)+1) & (ev'=1) & (start'=0) & (sm'=2)
            + 0.10: (counter'=mod(counter,50)+1) & (ev'=2) & (start'=0) & (sm'=2);
       //prob of rule satisfaction and violation
   [] sm=2 -> (sm'=0);
   [] sm=1 -> 0.10:(counter'= mod(counter,50)+1) & (ev'=3) & (start'=0) & (sm'=3)
            + 0.90:(counter'= mod(counter,50)+1) & (ev'=6) & (start'=0) & (sm'=3);
              //prob of anomaly or no anomaly
   [] sm=3 -> (sm'=0);
endmodule
```

Based on this model, we have allocated different probabilities for receiving a rule satisfaction, detecting an anomaly, as well as for resolving an anomaly, for each set of tests. Furthermore, we ran each of these sets five different times, each time by changing the number of sufficient conditions for issuing a certificate. More precisely, we run three different sets of tests with different probabilities, and for each set we changes the sufficiency conditions to 10, 20, 30, 40 and 50 rule satisfaction events. The probabilities that we checked were:

[1] Probability 90% to receive a rule satisfaction and 10% a rule violation, 10% to receive an anomaly event and 90% not to have an anomaly, and in case of an anomaly, 80% to resolve the anomaly and 20% not to resolve it;

[2] Probability 90% to receive a rule satisfaction and 10% a rule violation, 10% to receive an anomaly event and 90% not to have an anomaly, and in case of an anomaly, 90% to resolve the anomaly and 10% not to resolve it; and

[3] Probability 95% to receive a rule satisfaction and 5% a rule violation, 5% to receive an anomaly event and 95% not to have an anomaly, and in case of an anomaly, 95% to resolve the anomaly and 5% not to resolve it.

These probabilities were chosen only for testing and evaluation purposes, in order to check the behaviour of the life cycle model based on different percentages. In a real scenario we would expect a really small percentage for a security property rule violation, as this will revoke a certificate, and a small percentage also for an anomaly not to be resolve.

7.3.2.2 PRISM MODEL CHECKING RESULTS

For checking the correctness of the life-cycle model defined in the certification model, we first checked the probabilities to have a certificate with a violation (P=? [F (cert=1)&(nvio>0)]), and a certificate with an unresolved anomaly (P=? [F (cert=1) & (Unres>0)]). In all cases both probabilities were 0%, showing that there is no possibility to have either of the two cases, which was correct.

Further on, we checked the probability to have a resolved anomaly (resAnom>0), an unresolved anomaly (Unres>0), and no anomaly (noAnom>0) for each set of tests. These values should be the same for each set of tests concerning the probabilities, as they are only affected by the probabilities of sending each type of events. The results of each test are presented in the following table.

Table 30 - Probabilities for resAnom, Unres and noAnom

Probabilities	resAnom>0	Unres>0	noAnom>0
Sat- noAnom- ResAnom 90% - 90% - 80%	0.3999	0.1666	0.8823
Sat- noAnom- ResAnom 90% - 90% - 90%	0.4499	0.0900	0.8823
Sat- noAnom- ResAnom 95% - 95% - 95%	0.4749	0.0476	0.9470

As shown in Table 30, the probabilities of having a resolved or an unresolved anomaly, as well as having no anomaly at all (meaning that the assertion concerning an anomaly is not satisfied) depends only to the probabilities defined in the modules.

Finally, the last check that we have conducted was the different probabilities of having an issued or a revoked certificate, or an issued certificate with at least one anomaly, or one resolved anomaly, or no anomaly at all. In each test we were also changing the number of the sufficient condition events. The results are presented in the following table.

Table 31 - Probabilities of issuing certificates based on number of sufficient condition evidence

Probabilities	Number of Events	cert=1	cert=2	(cert=1)&(nan>0)	(cert=1)&(resAnom>0)	(cert=1)&(noAnom>0)
First set of experiments Sat- noAnom- ResAnom 90% - 90% - 80%	10	0.2844	0.2844	0.2174	0.2040	0.2843
	20	0.0812	0.0812	0.0722	0.0704	0.0812
	30	0.0232	0.0232	0.0220	0.0217	0.0232
	40	0.0066	0.0066	0.0064	0.0064	0.0066
	50	0.0018	0.0018	0.0018	0.0018	0.0018

Second set of experiments Sat- noAnom- ResAnom 90% - 90% - 90%	10	0.3139	0.3139	0.2470	0.2403	0.3139
	20	0.0989	0.0989	0.0899	0.0890	0.0989
	30	0.0312	0.0312	0.0300	0.0298	0.0312
	40	0.0098	0.0098	0.0096	0.0096	0.0098
	50	0.0031	0.0031	0.0030	0.0030	0.0030

Third set of experiments Sat- noAnom- ResAnom 95% - 95% - 95%	10	0.5828	0.5828	0.3993	0.3901	0.5828
	20	0.3401	0.3401	0.2726	0.2693	0.3401
	30	0.1985	0.1985	0.1737	0.1725	0.1985
	40	0.1159	0.1159	0.1068	0.1063	0.1159
	50	0.0676	0.0676	0.0643	0.0641	0.0676

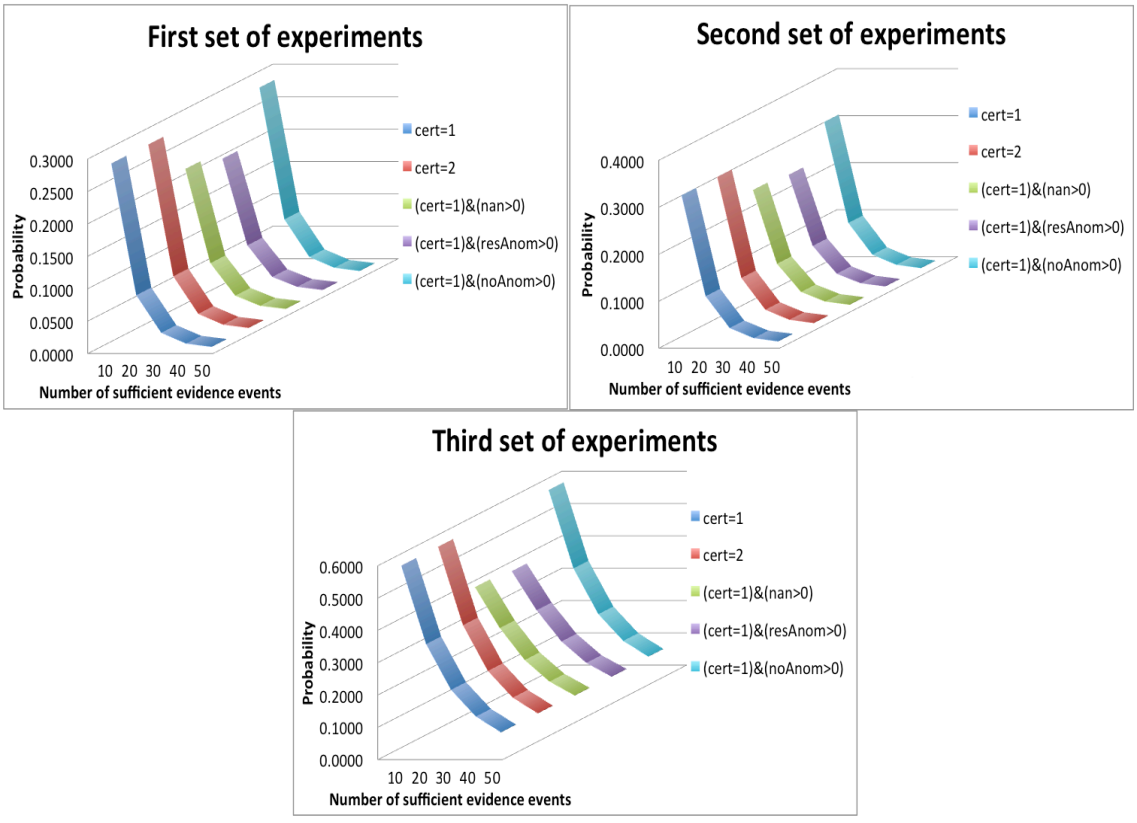


Figure 63 – Probabilities of issuing certificates based on number of sufficient condition evidence

As shown in Table 31 and in Figure 63, based on the different number of sufficient evidence and based on the different probabilities of each event to happen, the probabilities to have an issued certificate or a revoked certificate in every case is different. More precisely, when the number of sufficient evidence is increasing, meaning that more events of rule satisfaction are needed in order to issue a certificate, the probability to issue a certificate is decreasing. Similarly, when the probability to have a rule satisfaction increases or when the probability to resolve an anomaly increases, then the probability to issue a certificate also increases.

Furthermore, the probabilities to have an anomaly, or no anomaly or resolved anomaly whilst having also an issued certificate is also decreasing when more sufficient evidence are required.

Based on the results from the model checking, we can conclude that the life-cycle model is accurately defined in the certification model, as the behaviour of the model checking is realistic based on the events and form of the conducted tests.

7.3.2.3 THREATS TO VALIDITY

In this type of evaluation, there are several factors that can compromise the results. However, we tried to minimize these threats but still there are possibilities that the results might be biased.

The main impact of these experiments was the limited capability of the model checker tool to process many iterations of the defined model. More specifically, to compute each probability the Prism model checker process on average 900 iterations of the model. Though, when we tried to have more events for the sufficiency conditions, the tool could not process all possible iterations to calculate accurate probabilities, thus we limited our experiments to have up to 50 rule satisfaction events to be sufficient in order to issue a certificate. However, we expect that this would not be the case for a certification process of a real service.

Moreover, another constraint was the increased complexity of our model, in case we wanted to insert also a time automaton to express the expiry date of a certificate, which would lead to reissue a certificate. Thus, we did not include this factor in our model checking, even though in a real service certification process it is essential.

7.3.3 PERFORMANCE

In order to evaluate the performance of our monitoring-based certification approach, we have conducted an experiment based on a case study involving the certification of a real system. The system that we selected was the open source MySQL server [170], the RUBiS benchmark [196] to produce the workload, and an existing Protection Profile of Common Criteria.

Our choices for this type of experiment was influenced by:

- The complexity of this system,
- The existence of a Protection Profile generated by Oracle that specifies security properties for such systems based on Common Criteria [74] (aka Security Functional Requirements (SFR)), and
- The existence of benchmarks for creating realistic workloads for the MySQL server, which enable the evaluation of the automated certification process in realistic conditions. Moreover, since our approach does not support interventions with the purpose of addressing or restoring security violations, we focus only on the evaluation criteria of the MySQL server, based on the selected Protection Profile.

7.3.3.1 PERFORMANCE METHODOLOGY

The experiment that we set up to evaluate our approach realised a certification process for the security functional requirement FIA_UID.1.2 for the MySQL Server [74], based on a certification model including the assertions for defining the security property. We also used the RUBiS benchmark [196] to produce realistic workloads of events for the MySQL server and monitor the server for certification purposes during the execution of these workloads.

RUBiS is an auction site prototype, similar to the eBay, which implements the core functionality of an auction site, which allows users to browse for items, bid for items on sale, and pay for items from a wallet modelled after a bank account. To support its functionality, RUBiS implements a number of operations and transactions requiring varying levels of consistency and

isolation. To capture events (i.e., logs of queries) from the operation of the server, we used the MySQL AUDIT Plugin developed by McAfee [170]. The Audit plug-in is based upon the same technology as McAfee Database Activity Monitoring, allowing users to easily satisfy access full audit information. It may be used as a standalone audit solution or configured to feed data to external monitoring tools.

This Audit plug-in captures the logs created during the execution of the RUBiS workloads against the server. The events logged by the plugin were initially exported as .json files and subsequently parsed and converted into events, in the .xml format required by the framework. All the different systems used in our experiment, including RUBiS, MySQL, EVEREST and the proposed framework for monitoring-based certification process, were deployed on a cloud cluster involving a test-bed cloud cluster equipped with four 4-core server machines, each running at 2.20GHz, with 8GM of main memory, 450GB of disk space under Ubuntu 3.8.0.

The Common Criteria Protection Profile for the functional requirement FIA_UID.1.2 [74] refers to security requirements for database management systems in organisations where there are requirements for protection of the confidentiality, integrity and availability of information stored in the database. The main requirement is to provide basic database functionality, including allowing users to be granted the discretionary right to disclose the information to which they have legitimate access to other users.

The basic measure that we used in order to evaluate the performance of the certification process was the average time for making a decision about the monitoring assertion formulas in the model, called decision delay or d-delay. d-delay measures the difference between the time point when the latest event that is needed in order to make a decision about the satisfaction or otherwise of a monitoring formula occurs (tc) and the time when following the capture and processing of the event, the monitor makes a decision on whether the formula is satisfied (tp), i.e., d = tp– tc. Based on d- delay measures for individual instances of monitoring formulas, we calculated the average delay in the monitoring process using following formula:

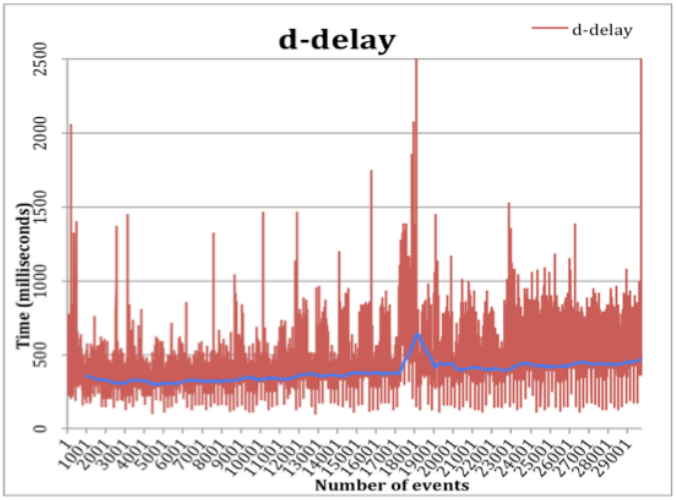
$$ave(d) = \sum d/N$$

Where:

- *d* is the d-delay of each monitoring rule instance, and
- *N* is the total number of monitoring rule instances for which a decision was made.

7.3.3.2 PERFORMANCE RESULTS

The graph in Figure 64 shows the d values for the different events of the RUBiS benchmark that caused monitoring rule checks in the certification model, and the moving average of d-delay (*ave(d)*) calculated over a window of 1000 events. The average value of d-delay across the whole RUBiS benchmark was 384.33 milliseconds (standard deviation = 118.92 milliseconds). As shown in the figure, *ave(d)* remained relatively stable throughout the execution of the benchmark, showing that certification results can be produced quickly following the actual events.



Events	Ave(d)
[1-5000]	326.02
[5001-10000]	324.47
[10001-15000]	357.59
[15001-20000]	443.31
[20001-25000]	413.87
[25001-29746]	443.83

Figure 64 – d-delay in execution of the database CM

In addition to the time needed to generate certification results, the execution of a monitoring-based certification model may have an impact on the operation of ToC as it is necessary to configure the ToC in order to produce the events needed for the monitoring process that underpins

the certification process. To evaluate this overhead in the case of the MySQL server, we executed two cases on the RUBiS benchmark:

- *Case (a)* without using the MySQL audit plugin; and
- *Case (b)* with the use of the MySQL audit plugin in the server.

The overhead was estimated by calculating the average throughout (i.e., the number of queries executed per minute) of the server in 10 different executions of case (a) and 10 different executions of case (b). Each of these 20 executions involved the execution of the same number of RUBIS queries against the server (~30,000 queries), but the queries executed in each execution were selected randomly by the RUBIS system. The completion of the execution of the different query sets took on average 18 minutes.

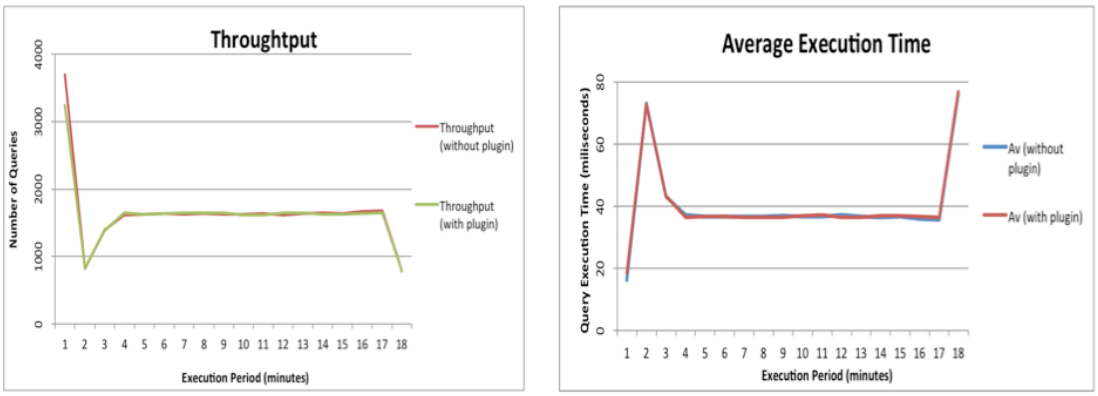


Figure 65 - Average throughput (i) and query processing time (ii) in executing the RUBIS benchmark on MySQL server with and without the MySQL AUDIT plugin

Table 32 - Average throughput and query execution time with and without the AUDIT plugin

Min	Throughput		Average Query Processing Time (msecs)	
	No Plugin	*With plugin*	*No plugin*	*With plugin*
1	3688.9	3245	16.27	18.49
2	819.9	824.7	73.18	72.75
3	1390.9	1386.1	43.14	43.29
4	1615	1651.1	37.15	36.34
5	1630.4	1629.7	36.8	36.82
6	1638.2	1636.5	36.63	36.66
7	1630.7	1645.9	36.79	36.45
8	1633.9	1643.9	36.72	36.5
9	1624.4	1648.5	36.94	36.4
10	1630.2	1619.9	36.81	37.04
11	1630.9	1611.8	36.79	37.23
12	1617	1646.9	37.11	36.43
13	1638.8	1648.5	36.61	36.4
14	1651.5	1627.3	36.33	36.87
15	1631.3	1628	36.78	36.86
16	1665.5	1635.4	36.03	36.69
17	1677.4	1649.8	35.77	36.37
18	788.2	779.6	76.12	76.96

The average throughput for cases (a) and (b) was measured per minute and the result is shown in the Throughput graph of Figure 65 and in details in Table 32. As shown in this graph the use of the MySQL AUDIT plugin had only a very minor effect on the performance of the server. The same is evident from Average Execution Time graph in Figure 65, which shows the average execution time per RUBIS query (in milliseconds), for every minute during the execution period. The absence of any significant effect is also evident, which shows the actual throughput and average query execution times for (a) and (b). The main difference in query execution time was observed only in the initial stage of the execution of each query set, when RUBiS sent queries to establish the connection to MySQL for each transaction thread.

7.3.3.3 THREATS TO VALIDITY

In this last type of evaluation experiment, there are different factors that can compromise our results. However, we tried to minimize them by running multiple iterations to cover as many possibilities as possible to be have more accurate and realistic results.

A main factor in this case is that the system that we monitored is not confirmed by other studies yet, with regards the overheads that might be introduced by monitoring its events. However, in order to have an adequate sample of evidence, we ran 20 different executions of the RUBiS benchmark and the MySQL Server, 10 of which were with the use of the plugin and 10 without it. Thus, based on the results we assumed to have an adequate sample of evidence to reach our conclusion.

Furthermore, all tests have been performed on a load-free system, so as to avoid any interference to the performance of the tests by the effect of concurrency.

Finally, in order to avoid introducing human errors in processing this large amount of data, all test results have been generated automatically, with the use of .json and .xml files as inputs to our translator and to our framework.

Chapter Eight

CONCLUSION AND FUTURE WORK

8.1 OVERVIEW

In this final chapter of the thesis, we provide an overview of the research work conducted regarding the monitoring-based certification framework for cloud services, which was presented in the earlier chapters. We also point out the main novelties of this framework and the contributions that our research has made to the state of the art. Moreover, the limitations of our research are also presented and directions for future work are discussed.

8.2 SUMMARY OF RESEARCH WORK

The framework designed and presented in Chapter 6 of this thesis was aimed at providing a novel basis for defining and automating the certification of security properties of cloud services. This process is based on continuous monitoring of the actual operations of cloud services, in order to gather the required evidence and enable a continuous assessment of the satisfaction of the security property of interest. The use of evidence coming from continuous monitoring provides coverage of contextual conditions that might not be possible to envisage, test or simulate through other forms of assessment of security properties, such as through testing or static analysis, and therefore provides, in our view, an advantage over them.

To continuously monitor the events, we used the *EVEREST* monitoring tool. To specify an automatically executable certification process, we have defined an XML based language for specifying monitoring based certification models, as presented in Chapter 4. This language can be used to define all the necessary information regarding the certification process for a specific target of certification. More specifically, as required by the schema, a Certification Model contains

information regarding the Target of Certification (ToC) that needs to be certified, the security property that the certificate refers to, the way of assessing the defined security property, and all relevant conditions that need to be met in order to generate and issue a monitoring-based certificate.

In order to define the security property assertions, a new language was defined based on the Event Calculus language. Furthermore, a translator was developed to translate the assertions of the Certification Model into the *EC- Assertion* language, which is the language that the *EVEREST* monitoring tool uses to define monitoring rules and assumptions. The extensions in the language were made to support the definition of complex security properties assertions.

8.3 CONTRIBUTIONS

The main contributions of this research are summarised below.

- *Development of an XML based language for expressing monitoring based certification models*

  We have defined a new XML based language in order to be able to specify the monitoring based certification models.. This language is defined by an XML schema and enables the automated specification and realization of certification models for an objective assessment of security properties, using evidence gathered through continuous monitoring. The XML schema of the language was presented and explained in Chapter 4 and examples of defined models were given in Chapter5.

  The purpose of such models is to define: (i) the security property that needs to be certified, (ii) the types and extent of evidence that should be acquired in order to be able to certify the property, (iii) the life cycle of certificates of the given type, and (iv) the agents which will have the responsibility to carry out different parts of the process.

  Since, in the monitoring based certification process the evidence required for assessing and verifying security properties is acquired through continuous monitoring of the cloud service operations, the evidence underpinning certificates of this type can cover contextual

conditions that might not be possible to predict, test or simulate through other forms of assessment, such as testing or static analysis, that take place before the deployment of a cloud service.

- *Definition of an XML based Assertion Language to express the assertions of the security property*

  In order to enable the formal specification of security properties that can drive the monitoring process as part of the certification model, we have developed a new XML language. This language was introduced in Section 4.3.1.4 and has a similar semantic foundation to the EC-Assertion language of EVEREST, with respect to the specification of monitoring conditions in terms of events and fluents and first order temporal logic formulas of Event Calculus. However, the new language provides higher level and aggregate syntactic constructs, in order to specify more complex security property assertions that were needed for the work done in this research. It has also introduced the concept of "executable" events, whose purpose is to perform complex computations that may be needed during the monitoring process.

  Following the introduction of this new language, it became necessary to introduce a new component to our framework, which could support the translation of these assertions of the certification model, into the operational monitoring language of the EVEREST monitoring tool that we used.

  In particular, as discussed in Section 6.3.2.3, the new component is responsible to translate the assertions for a security property as specified in the certification model, into the EC-Assertion language of the EVEREST monitoring tool.

- *Development of a framework to process the Certification Models and automatically generate monitoring-based certificates.*

  Having defined a certification model and a language to express the security properties assertions, we then developed a framework to support the monitoring based certification process, based on the information defined in the model. As described in Chapter 6, this framework allows the processing of the defined certification models and the generation and

management of monitoring-based certificates. More specifically, it is able to translate the security property assertions, defined in the certification model, to the EC-Assertion language that the monitor understands, and to process all conditions defined in the certification model, as well as the monitoring results received by the monitor, in order to decide and change the status of the certificate based on this information. For the decision of the status of the certificates, the framework process the life-cycle model defined in the certification model, following the algorithm presented in Section 6.3.4.4.

The developed framework is novel with respect to the existing state of the art. More specifically, existing cloud certification and other schemes do not support the continuous monitoring of the resources in order to acquire the necessary evidence to certify a service, as we discussed in Chapter 2. Moreover, they are not able to automatically identify appropriate evidence acquisition plans and security assessment models. The proposed framework fills these gaps, by supporting the monitoring of security properties and resources, tailored to the need of users, in order to provide adequate monitoring evidence for the purposes of certification. Furthermore, it can support the expression and automated execution of other aspects of the process, including the specification and verification of evidence sufficiency conditions and the specification and execution of life cycle models for certification.

- *Evaluation of the approach*

Three different activities were conducted in order to evaluate the outcomes of this research. These were: (i) a subjective evaluation of the comprehensiveness and complexity of the proposed certification models, (ii) a formal analysis and verification of the life cycle models defined in certification models, and (iii) an experimental evaluation of the operational correctness and performance of the proposed framework.

The first activity was based on interactive sessions with experts in the area of certification and on a questionnaire that the participants had to answer, in order to give their feedback about the complexity and comprehensiveness of the certification models used in our approach. According to the results of this evaluation, it showed that the proposed model is capable to accurately specify the conditions to support the continuous monitoring certification process for assessing most security properties of cloud services.

The second activity focused on the use of model checking tool to verify the life cycle model specified in the proposed certification model. This analysis was based on a symbolic analysis and a model checking using the PRISM model checker. The results of this activity proved that the life-cycle model is able to accurately support the certification process.

The third activity was based on the use of a certification model for database management systems (DBMS) as part of an e-commerce system, which was established by a benchmark used for performance evaluation and on a Protection Profile of Common Criteria. This activity showed that certification does not affect the performance of the system that is being certified (i.e., the TOC), as measured by different metrics (e.g., TOC throughput, TOC transaction execution time).

8.4 LIMITATIONS

The framework that we have developed for the monitoring based certification process of cloud services has some limitations.

The main limitation is that the framework is able to process only one certification model at a time. Even though we use the Certification Model ID to all the objects used by the process (e.g. rules, assumptions, evidence, etc.) in order to differentiate the execution of different monitoring processes, the EVEREST monitor that we used to support the monitoring process is not able to handle multiple different assertions.

Furthermore, our approach requires a specific XML format for the received events in order for the monitor to process them and for the framework to store them for auditing purposes. Thus, it requires from the service providers to adjust the format of their produced events, based on the expected format, and to include all needed information in the relevant tabs of the event XML schema.

Finally, in order to correctly define a Certification Model, as presented in Chapter 4 and 5, some training will be needed, especially for the definition of the security properties assertions.

8.5 FUTURE WORK

This research proposed a framework that allows the certification of cloud services, based only on continuous monitoring. However, this approach can be further used to provide a higher level of certification, by combining it with other ways of evidence collection. Therefore, some directions for future work are listed below:

- ***Hybrid Certification***

In some cases the use of monitoring based certification is not fully sufficient for achieving a high degree of assurance. For example, in the case of certifying the authorisation security property of a service, where only authorised users can modify resources, the users request events for updating data can be monitored and then trigger a testing agent to verify the users authorisation rights to execute this operation. Another example of combining evidence collection techniques could be used to certify the protection of a system from SQL injection attacks (integrity security property). In this case a testing could be performed in a pre-production environment and then in a production environment, where the TOC should not be threatened by tests, instead of attacking the TOC the test-base certification process can rely on monitoring evidence, checking that no query containing SQL injection is executed.

Therefore, in such cases the combination of different types of evidence (e.g., monitoring and testing) may be useful in achieving the required assurance level. The combination of such types of evidence has been termed as "hybrid certification" in [131]. As defined in [131], hybrid certification process aims to combine evidence from multiple sources of different nature (such as Monitoring, Testing, or Trusted Computing (TC) Proofs) in order to determine the satisfaction of a security property by a service. Such a combination permits to develop more consistent and concrete processes, which can overcome limitations introduced by certification processes based on a single way of collecting evidence. Hybrid certification can be achieved in two modes, depending on the module in charge of combining the evidence coming from different sources:

a) In a *dependent* mode, where the certification process is driven by one of the mechanisms used for evidence collection. The driving mechanism, as required by the

conditions specified in the certification model, it triggers the dependent one as a subordinate in order to generate the additional evidence; and

b) In an *independent* mode, where two evidence collection processes are executed independently and a third module collects the evidence from these two processes. This module evaluates the evidence coming from the different sources, as specified in the certification model, in order to assess the validity of the property.

Whilst the first of the above modes has been realised by existing work [69], the second needs to be implemented and our approach could be extended to support it.

- ***Incremental Certification***

To support the dynamic behaviour of the cloud, the certification process should be able to dynamically certify and constantly verify the validity of a certificate, at runtime. The incremental certification process aims to provide such ability, by avoiding as much as possible the re-certification of a service in case a change occurs (i.e. a possible change of the Virtual Machine used, service migration, or changes to the configurations of platform services). This can be achieved by adapting the certification process to be able to reuse available evidence that is still sufficient for the certification and partially re-evaluate parts of the service that has changed. Furthermore, previous issued certificates can also be reused as evidence for generating new ones, which refer to the same security property and cloud service.

However, the monitoring-based certification process is considered to be an incremental certification, as it continuously checks for a property to be valid. In particular, the deployment on a different stack it can automatically be achieved by moving the event captors in the new stack.

- ***Certificate composition***

Certificate Composition is a concept that has already been introduced in SOA environment in [16][189][190][188]. This type of certification aims to reuse the evidence in existing certificates

of component services, in order to issue a new certificate for a composite service. However, in the work done for this type of certification, the process does not support a specific description in the form of a certification model, thus a composition of this type requires certification authorities to be involved and a time consuming selection of candidates based on existing evidence. This work could be enhanced by a simpler and effective approach that can exploits the concept of the certification model introduced in this research.

Extending the work presented in this thesis to support composition of certificates would require the ability to assess the compatibility of the security properties expressed in the relevant monitoring based certificates, and the compatibility and complementarity of the monitoring evidence underpinning them.

REFERENCES

[1] "Amazon CloudWatch", http://aws.amazon.com/cloudwatch/

[2] "Blue Vector Monitoring and Management Engine for Cloud Computing", White paper, Blue Vector Systems, www.bluevector.com

[3] Abramowski D., "Monitoring Applications in the Cloud", March 10, 2009, SYS-CON Media, Inc.

[4] Accorsi R. and Lowis L. "ComCert: Automated Certification of Cloud-based Business Processes." ERCIM News 2010(83), pp. 50-51, 2010.

[5] Accorsi R., Lowis L., Sato Y. "Automated Certification for Compliant Cloud-based Business Processes, Business & Information Systems Engineering", 2011

[6] Aceto G., Botta A., De Donato W., Pescapè A. "Cloud Monitoring: definitions, issues and future directions", 1st IEEE International Conference on Cloud Networking (IEEE CloudNet'12)", Paris (France), pp. 63-67, 2012.

[7] Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), http://www.ssi.gouv.fr/fr/certification-qualification/cesti/, 2011

[8] Agostini R., Bettini C., Cesa-Bianchi N., Riboni D., Ruberl M. and Sala C. "Towards highly adaptive services for mobile computing". In Proceedings of IFIP TC8 Working Conference on Mobile Information Systems (MOBIS), Springer, 2004, pp. 121-134.

[9] Al-Moayed A. and Hollunder B. Quality of service attributes in web services. In Proc. of the 5th International Conference on Software Engineering Advances (ICSEA 2010). Nice, France, August 2010.

[10] Alves A., Arkin A., Askary S., Barreto C., Bloch B., Curbera F., Ford M., Goland Y., Guizar A., Kartha N., Liu C.K., Khalaf R. "Web services business process execution language version 2.0. OASIS Standard", 2007.

[11] Andreozzi S., Bortoli N.D., Fantinel S., Ghiselli A., Rubini G.L., Tortone G. and Vistoli M.C. "GridICE: a monitoring service for grid systems," Future Generation Computer Systems, 21 (4), pp. 559–571, April 2005.

[12] Anisetti M. et al., "ASSERT4SOA: Toward Security Certification of Service-Oriented Applications", OTM 2010 Woekshops, pp. 38-40, 2010.

[13] Anisetti M., Ardagna C.A. and Damiani E. "A Low-Cost Security Certification Scheme for Evolving Services", In Proc. Of IEEE 19th International Conference on Web Services, pp. 122-129, 2012.

[14] Anisetti M., Ardagna C.A. and Damiani E. "Defining and matching test-based certificates in open SOA". In Proc. of the Second International Workshop on Security Testing (SECTEST 2011). Berlin, Germany, March 2011, short paper.

[15] Anisetti M., Ardagna C.A. and Damiani E. "Fine-grained modeling of web services for test-based security certification". In Proc. of the 8th International Conference on Service Computing (SCC 2011), Washington, DC, USA, July 2011.

[16] Anisetti M., Ardagna C.A., Damiani E. and Maggesi J. “Security certification-aware service discovery and selection”. In Proceedings of the Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1-8, 2012.

[17] Ardagna C.A., Asal R., Damiani E., and Vu Q.H. “On the management of cloud non-functional properties: The cloud transparency toolkit”. In Proceedings of IFIP NTMS 2014, 2014.

[18] Ardagna C.A., Asal R., Damiani E., and Vu Q.H.. “From Security to Assurance in the Cloud: A Survey”. ACM Computing Surveys, 48(1), Article 2, 2015.

[19] Armando A. and Compagna L. SATMC: a SAT–based model checker for security protocols. In Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004), volume 3229 of Lecture Notes in Computer Science. Springer Verlag, pp 730-733, 2004.

[20] Armando A., Carbone R., Compagna L., Cuellar J., and Tobarra L.. Formal analysis of SAML 2.0 web browser single sign–on: breaking the SAML–based single sign–on for google apps. In Proceedings of the 6th ACM workshop on Formal methods in security engineering ,ACM New York, NY, USA, pp. 1–10, 2008.

[21] Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I. and Zaharia M. “Above the Clouds: A Berkeley Review of Cloud Computing”, In Tech. Rep. UCB/EECS 2009, EECS Department, U.C. Berkeley, 2009.

[22] Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I. and Zaharia M. “A view of cloud computing”. CACM 53, 4, pp. 50-58, 2010.

[23] ASSERT4SOA Project. ASSERT4SOA, Advanced Security Service cERTificate for SOA. www.assert4soa.eu, 2011.

[24] AVANTSSAR. Research project IST–2001–39252. http://www.avantssar.eu/.

[25] AVISPA. Research project IST–2001–39252. http://www.avispa–project.org/.

[26] Bacic E.M. The canadian trusted computer product evaluation criteria. In Proc. of the Sixth Annual Computer Security Applications Conference, Tucson, AZ, USA, December 1990.

[27] Baiardi F., Cilea D., Sgandurra D., Ceccarelli F. “Measuring Semantic Integrity for Remote Attestation”. In Proceedings of the 2nd International Conference on Trusted Computing, April 06-08, 2009, Oxford, UK [doi>10.1007/978-3-642-00587-9_6]

[28] Baresi L. and Di Nitto E. “Test and Analysis of Web Services”. Springer, New York, USA, 2007

[29] Baresi L. and Guinea S. “Dynamo: Dynamic Monitoring of WS-BPEL Processes”, In Proccedings of the 3rd International Conference On Service Oriented Computing (ICSOC 05), Amsterdam, pp. 478-483, 2005.

[30] Baresi L., Guinea S. and Plebani P. “WS-Policy for Service Monitoring” In Technologies for E-Services, 6th International Workshop, TES 2005, pp. 72–83, 2005.

[31] Barham P. et al. "Xen and the art of virtualization." In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03), 37(5), pp. 164-177, 2003.

[32] Baset A.S., Wang L. and Tang C. “Towards an understanding of oversubscription in cloud”. In Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services , Hot-ICE’12, pp. 7–7, Berkeley, CA, USA, USENIX Association, 2012.

[33] Basili V.R., Caldiera G., Rombach H.D. “The Goal Qiestion Metric Approach”. In Encyclopedia of Software Engineering, J. Marciniak (Ed), Wiley, 1994.

[34] Berger S., Caceres R., Goldman K.A., Perez R., Sailer R. and Van Doorn L. “vTPM: Virtualizing the trusted platform module”. In Proceedings of USENIX-SS, 2006.

[35] Bettini C., Maggiorini D. and Riboni D. Distributed Context Monitoring for the Adaptation of Continuous Services. World Wide Web, 10(4):503-528, 2007.

[36] Bianculli D. and Ghezzi C. “Monitoring Conversational Web Services”. In IWSOSWE’ 07, 2007.

[37] Blake M.B., Kahan D.R. and Nowlan M.F. Context Aware Agents for User-oriented Web Services Discovery and Execution. Distrib. Parallel Databases, 21(1), pp. 39-58, 2007.

[38] Boampong P.A. and Wahsheh L.A. “Different facets of security in the cloud”. In Proceedings of CNS 2012, 2012.

[39] Bowels K., Juels A. and Oprea A. “HAIL: a high-availability and integrity layer for cloud storage”, In Proceedings of the 16th ACM conference on Computer and Communications Security (CCS ’09), 2009.

[40] Burton S., Kaliski J. and Pauley W. “Toward risk assessment as a service in cloud environments”. In Proccedings of the 2nd USENIX conference on Hot topics in cloud computing, pp. 13–13, Berkeley, CA, USA, June 2010.

[41] Cachin C., Keider I., and Shraer A., “Trusting The Cloud”, IBM Research, Zurich Research laboratory, 2009.

[42] Canfora G. and Di Penta M. Service-oriented architectures testing: A survey. Software Engineering: International Summer Schools, pp. 78–105, 2009.

[43] Carlsson C. and Fullér R. “Predictive Probabilistic and Possibilistic Models Used for Risk Assessment of SLAs in Grid Computing”, Information Processing and Management of Uncertainty in Knowledge-Based Systems 13th International Conference (IPMU 2010), Germany, Part II, Vol.81, pp. 747-757, 2010.

[44] Catteddu D. and Hogben G., “Cloud Computing: Benefits, Risks and Recommendations for Information Security.”, European Network and Information Security Agency (ENISA), 2009.

[45] Celesti A., Tusa M. and Puliafto A., Security and Cloud Computing:InterCloud Identity Management Infrastructure, In Procceedings of the 19th IEEE International Workshops on Enabling Technologies:Infrastructures for Collavorative Enterprises, pp. 263-265, 2010.

[46] Certification Commission for Healthcare Information Technology. Certification Handbook CCHIT Certified® 2011 Certification Program, May 2011. http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Certification%20Handbook%20May%2017%202011%20FINAL_1.pdf

[47] Cetinkaya E.K. and Sterbenz J.P.G. “A Taxonomy of Network Challenges”. In Proceedings of the 9th IEEE/IFIP International Conference on the Design of Reliable Communication Networks (DRCN), pp. 322–330, Budapest, 2013.

[48] Chahal S., Steichen J.H., Kamhout D., Kraemer R., Li H. and Peters C. “An Enterprise Private Cloud Architecture and Implementation Roadmap”. IT@Intel White Paper, Intel Information Technology, June 2010. Available at: http://www.intel.co.uk/content/dam/doc/guide/intel-it-enterprise-cloud-architecture-roadmap-paper.pdf [retrieved June 2013]

[49] Chandran S.P. and Angepat M. “Cloud Computing: Analysing the Risks involved in Cloud Computing Environments”, In Proceedings of Natural Sciences and Engineering, Sweden, 2010.

[50] Chen Y., Paxson V., and Katz R.H. “What’s New About Cloud Computing Security?”, University of California, Berkeley Report No. UCB/EECS-2010-5, 20(2010), pp. 2010-5, 2010.

[51] Chevalier Y., Compagna L., Cuellar J., Drieslma P.H., Mantovani J., Mdersheim S., and Vigneron L. “A high level protocol specification language for industrial security–sensitive protocols”. In Workshop on Specification and Automated Processing of Security Requirements (SAPS 2004), 2004.

[52] Chow R., Golle P., Jakobsson M., Shi E., Staddon J., Masuoka R. and Molina J. “Controlling data in the cloud: outsourcing computation without outsourcing control”. In Proc. of the 2009 ACM workshop on Cloud computing security, pp. 85-90, 2009.

[53] Chow R., Jakobsson M., Masuoka R., Molina J., Niu Y., Shi E. and Song Z. “Authentication in the clouds: a framework and its application to mobile users”. In Proceedings of the 2010 ACM workshop on Cloud computing security workshop pp. 1-6, 2010.

[54] Clayman S. et al., “Monitoring Service Clouds in the Future Internet”. In Towards the Future Internet - Emerging Trends from European Research, (eds.) Tselentis, G et al., IOS Press: Amsterdam, pp. 115 – 126, 2010.

[55] Cloud Industry Forum. http://www.cloudindustryforum.org/about-us

[56] Cloud Security Alliance (CSA), https://cloudsecurityalliance.org/research/ctp/

[57] Cloud Security Alliance, “The notorious nine: Cloud computing top threats in 2013”, v.1.0, Cloud Security Alliance, February 2013, available from: http://cloudsecurityalliance.org/research/top-threats/ [retrieved: April 2014]

[58] Cloud Security Alliance, Cloud Audit, Available from: https://cloudsecurityalliance.org/research/cloudaudit/

[59] Cloud Security Alliance, Cloud Controls Matrix, Available from: https://cloudsecurityalliance.org/research/ccm/

[60] Cloud Security Alliance, CSA Security, Trust and Assurance Resigtry (STAR), Avaliable from: https://cloudsecurityalliance.org/star/

[61] Cloud Security Alliance, Security Guidance for Critical Areas of Focus in Cloud Computing vol. 2, available from: http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf [retrieved: June, 2013].

[62] COBIT, http://www.isaca.org

[63] Coker G. et al., “Principles of remote attestation”. From the issue entitled "Special Issue:10th International Conference on Information and Communications Security (ICICS)".

[64] ComCert, http://www.telematik.uni-freiburg.de/comcert

[65] Common Criteria (CC) for Information Technology Security Evaluation, CCDB USB Working Group, 2012, part 1-3. Available from: http://www.commoncriteriaportal.org..

[66] Compagna L. “SAT–based Model–Checking of Security Protocols”. PhD thesis, Universit degli Studi di Genova and the University of Edinburgh, 2005.

[67] Comuzzi M. and Spanoudakis G. "Dynamic Set Up of Monitoring Infrastructures for Service Based Systems", In Proceedings of the 25th Annual ACM Symposium on Applied Computing, Track on Service Oriented Architectures and Programming, pp. 2414-2421, March 2010.

[68] CUMULUS Project, Deliverable “D2.1 Security-aware SLA specification language and cloud security dependency model v1.01”, September 2013. Available from http://www.cumulus-project.eu/ [retrieved 2013]

[69] CUMULUS Project, Deliverable “D2.4 Final CUMULUS Certification Models”, May 2015. Available from http://www.cumulus-project.eu/ [retrieved 2015]

[70] CUMULUS Project, Deliverable “D6.3 Smart Cities Pilot”, December 2014. Available from http://www.cumulus-project.eu/ [retrieved 2014]

[71] CUMULUS Project, Deliverable “D6.4 e-Health Pilot”, December 2014. Available from http://www.cumulus-project.eu/ [retrieved 2015]

[72] Damiani E. and Mana A. “Toward WS-Certificate”, In Proceedings of the ACM Workshop on Secure Web Services (SWS 2009), pp. 1-2, 2009.

[73] Damiani E., Ardagna C.A. and Ioini N.E., "Open Source Security Certification". Springer, December 2008.

[74] Database Management System Protection Profile, Issue 2.1, May 2000, Available from http://www.commoncriteriaportal.org/files/ppfiles/T129%20- %20PP%20v2.1%20%28dbms.pp%5B1%5D%29.pdf

[75] Dempsey K. et al., Information Security Continuous Monitoring (ISCM) for Federal Systems and Organisations, NIST 800-137, 2011, available from: http://csrc.nist.gov/publications/nistpubs/800-137/SP800-137-Final.pdf

[76] Dimitrakos T. “The BEinGRID Project”, In Grid and Cloud Computing A Business Perspective on Technology and Applications, Springer Berlin Heidelberg, 2009.

[77] Djemame K., Armstrong D.J., Kiran M., Jiang M. “A Risk Assessment Framework and Software Toolkit for Cloud Service Ecosystems”. In Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 119-126, 2011.

[78] DoD, Department Of Defense Trusted Computer System Evaluation Criteria. USA Department of Defence, December 1985. http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt

[79] Doelitzscher F., Reich C., Knahl M.H. and Clarke N.L. "Incident detection for cloud environments", Proc. of the Third International Conference on Emerging Network Intelligence,2011, pp. 100-105.

[80] Doelitzscher F., Reich C., Knahl M.H., Passfall A. and Clarke N.L., "An agent based business aware incident detection system for cloud environments", Journal of Cloud Computing: Advances, Systems and Applications, 1(9), 2012.

[81] Dong W.L. and Yu H. “Web service testing method based on fault-coverage”. In Procceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 2006). Hong Kong, China, October 2006

[82] Dranidis D., Kourtesis D., Ramollari E. “Formal Verification of Web Service Behavioural Conformance through Testing”. In Proc. of the 3rd South-East European Workshop on Formal Methods, November 2007

[83] Dranidis D., Ramollari E., Kourtesis D. “Run-time Verification of Behavioural Conformance for Conversational Web Services”. European Conference on Web Services, Eindhoven, November 2009

[84] Emeakaroha V.C., Calheiros R.N., Netto M.A.S., Brandic I. and De Rose C.A.F. “DeSVi: An Architecture for Detecting SLA Violations in Cloud Computing Infrastructures” 2nd International ICST Conference on Cloud Computing, 2010.

[85] Emeakaroha V.C., Ferreto T., Netto M.A.S., Brandic I., De Rose C.A.F. “Casvid: Application Level Monitoring for SLA Violation Detection in Clouds” In Proccedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference (COMPSAC 2012), pp. 499 – 508, 2012.

[86] Emeakaroha V.C., Netto M.A.S., Calheiros R.N., Brandic I., Buyya R., De Rose C.A.F. “Towards autonomic detection of SLA violations in Cloud infrastructures”. Future Generation Comp. Syst. 28(7), , pp. 1017-1029 2012.

[87] ENISA, Security Certification Practice in the EU: Information Security Management Systems – A Case Study, v1, October 2013, available from: https://www.enisa.europa.eu/

[88] EuroCloud Deutschland eco e.V., “Requirements”. http://www.saas-audit.de/en/511/requirements/

[89] European Network and Information Security Agency (ENISA), “Cloud Computing: Benefits, Risks and Recommendations for Information Security”, (Eds) L. Dupre and T. Haeberlen, 2012. Available at: https://resilience.enisa.europa.eu/cloud-security-and-resilience/publications/cloud-computing-benefits-risks-and-recommendations-for-information-security [retrieved 2013]

[90] Fally B., Ahtes J. and Martensmeier T. “Approaching the cloud: Better business using Grid solutions, Twenty-five successful case studies from BEinGRID”, BEinGRID Consortium, 2010.

[91] Federal Information Security Management. https://www.dhs.gov/federal-information-security-management-act-fisma

[92] FedRAMP Office, Guide to Understanding FedRAMP. Version 1.2, 22 April 2013, p. 51. Available from: http://www.gsa.gov/portal/mediaId/170599/fileName/Guide_to_Understanding_FedRAMP_042213

[93] Feng J., Chen Y. and Summerville D.H. “A Fair Multi-Party Non- Repudiation Scheme for Storage Clouds”, Int. Conf. on Collaboration Technologies and Systems (CTS), pp. 457-465, May 2011.

[94] Ferrer A. J., Hernández F., Tordsson J., Elmroth E., Ali-Eldin A., Zsigri C., Sirvent R., Guitart J., Badia R.M., Djemame K., Ziegler W., Dimitrakos T., Nair S.K., Kousiouris G., Konstanteli K., Varvarigou T., Hudzia B., Kipp A., Wesner S., Corrales M., Forgó N., Sharif T., and Sheridan C. “OPTIMIS: A holistic approach to cloud service provisioning”, Future Generation Computer Systems, 28(1), pp. 66-77, 2012.

[95] Foster H. and Spanoudakis G. "Advanced Service Monitoring Configurations with SLA Decomposition and Selection", In Proceedings of the 26th ACM Symposium Applied Computing – Track on Service Oriented Architecture and Programming, , pp. 1582-1589, March 2011.

[96] Foster H. and Spanoudakis G. "Dynamic Creation of Monitoring Infrastructures". In Service Level Agreements for Cloud Computing. Vol. 3, Wieder, P.; Butler, J.M.; Theilmann, W.; Yahyapour, R. (Eds.), Springer, 2011.

[97] Foster H. and Spanoudakis G. "SMaRT: A Workbench for Reporting the Monitorability of Services from SLAs", In Proccedings of the 3rd International Workshop on Principles of Engineering of Service-Oriented Systems – in conjunction with the 33rd ACM/IEEE International Conference on Software Engineering, pp.36-42, 2011.

[98] Frantzen L., Tretmans K. and De Vries E. "Towards model-based testing of web services". In Proc. of the International Workshop on Web Services - Modeling and Testing – WS-MaTe 2006. pp. 67–82. Palermo, Italy, June 2006.

[99] Fraunhofer Institute for Secure Information Technology SIT, Darmstadt. Simple Homomorphism Verification Tool –Manual, 2007.

[100] Gan Y., Chechik M., Nejati S., Bennett J., O'Farrell B. "Runtime monitoring of web service conversations", In Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, Richmond Hill, Ontario, Canada , 2007.

[101] Ganglia, 2011. http://ganglia.sourceforge.net/ [retrieved April 2012)

[102] Ghezzi C. and Guinea S. "Runtime Monitoring in Service Oriented Architectures", In Test and Analysis of Web Services, (Eds) Baresi L. & di Nitto E., Springer, pp. 237-264, 2007.

[103] Gilani W. et al. "SLA-aware Service Management", Deliverable DA3.a, M38, F7 SLA@SOI Project, 2011, available from: http://sla-at-soi.eu/wp-content/uploads/2011/08/D.A3a-M38-SLA-aware-service-management.pdf [retrieved December 2011]

[104] Grabner A. "Challenges of Monitoring, Tracing and Profiling your Applications running in The Cloud", May 06, 2009, SYS-CON Media, Inc.

[105] Grabner A. "Proof of Concept: dynaTrace provides Cloud Service Monitoring and Root Cause Analysis for GigaSpaces", May 06, 2009, SYS-CON Media, Inc.

[106] Gridipedia, 2011. SLA Monitoring and Evaluation Technology Solution http://www.gridipedia.eu/sla-monitoring-evaluation.html (last accessed on 27/12/2011)

[107] Grobauer B., Walloschek T., Stocker E. "Understanding Cloud Computing Vulnerabilities", IEEE Security & Privacy, 9(2), pp. 50-57, 2011.

[108] Guergens S. and Rudolph C. "Security Analysis of (Un–) Fair Non–repudiation Protocols". Formal aspects of computing, 2004.

[109] Guergens S., Ochsenschlaeger P., Rudolph C. "Role based specification and security analysis of cryptographic protocols using asynchronous product automata". In DEXA 2002 International Workshop on Trust and Privacy in Digital Business. IEEE, 2002.

[110] Guo, Z. "A Governance Model for Cloud Computing", International Conference on Management and Service Science (MASS), pp. 1-6, 2010.

[111] Gurgens S., Rudolph C. and Vogt H. “On the Security of Fair Non- Repudiation Protocols”, International Journal of Information Security, 4(4), pp. 253- 262, 2005.

[112] Haeberlen A. “A case for the accountable cloud.”, SIGOPS Oper. Syst. Rev. 44(2): 52-57, April 2010.

[113] Halle S. and Villemaire R. "Runtime Monitoring of Message-Based Workflows with Data", In Procceding of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC ’08), 2008, pp. 63-72.

[114] Hallé S. and Villemaire R. "Runtime monitoring of web service choreographies using streaming XML", I n Proceedings of the 2009 ACM symposium on Applied Computing (SAC '09). ACM, New York, pp. 2118-2125, 2009.

[115] Hanna S. and Munro M. An approach for specification-based test case generation for web services. In: Proc. of the IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007). Amman, Jordan, May 2007

[116] Heckel R. and Lohmann M. “Towards contract-based testing of web services” In Proccedings of the International Workshop on Test and Analysis of Component Based Systems(TACoS 2004). Barcelona, Spain, pp. 517-526, March 2004.

[117] Heiser J. and Nicolett M. “Assessing the Security Risks of Cloud Computing”, Gartner technical report, June 2008.

[118] Herrmann D.S. “Using the Common Criteria for IT security evaluation”. Auerbach Publications, 2002.

[119] Hua Z., Gong B. and Xu X. “A DS-AHP approach for multi-attribute decision making problem with incomplete information”. Expert Systems with Applications, 34(3), pp. 2221–2227, 2008.

[120] Hudic A., Krotsiani M., Tauber M., Spanoudakis G., Lorünser T., Mauthe A. and Weippl E.R.R. “A Multi-Layer and Multi-Tenant Cloud Assurance Evaluation Methodology”, In Proccedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014), Singapore, 2014.

[121] ICSA Labs. Testing Services. https://www.icsalabs.com/testing-services, 2011

[122] ISO/ IEC 27005:2007, Information Technology—Security Techniques—Information Security Risk Management, International Organisation Standardization, 2007.

[123] Jahl C. “The information technology security evaluation criteria”. In Proccedings of the 13th International Conference on Software Engineering, Austin, TX, USA, pp. 306 - 312, May 1991.

[124] Jank K. “Reference Architecture”. Adaptive Services Grid Deliverable D6.V-1, 2005.

[125] Jensen M., Schwenk J., Gruschka N. and Iacono L.L. “On Technical Security Issues in Cloud Computing”, In Proc. of the IEEE International Conference on Cloud Computing, 2009, pp. 109-116.

[126] Jin S., Ahn J., Cha S., and Huh J. "Architectural Support for Secure Virtualization under a Vulnerable Hypervisor", In Procceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44), pp. 272-283, December, 2011.

[127] Jokhio M., Dobbie G. and Sun J. “Towards specification based testing for semantic web services”. In Proc. of the 20th Australian Software Engineering Conference (ASWEC 2009). Gold Coast, Australia, April 2009.

[128] Jurjens J. UMLsec: Extending UML for secure systems development. Lecture Notes in Computer Science, pages 412–425, 2002.

[129] Kamara S. and Lauter K. “Cryptographic cloud storage”, 14th International Conference on Financial cryptograpy and data security, pp. 136-149, 2010.

[130] Kang S. et al. “Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments”. In Proc. of the 6th international conference on Mobile systems, applications, and services (MobiSys '08), ACM, New York, pp. 267-280, 2008.

[131] Katopodis S., Spanoudakis G., and Mahbub K. “Towards hybrid cloud service certification models”. InProceedings of the IEEE International Conference on Services Computing (SCC), pp. 394-399, 2014.

[132] Katsikas S.K., Lopez J. and Pernul G., “Trust, Privacy and Security in E-business: Requirements and Solutions”, In Proceedings of the 10th Panhellenic conference on Advances in Informatics (PCI'05), Bozanis, P. and Houstis, E.N. (Eds.), Springer-Verlag, Berlin, Heidelberg, pp. 548-558, 2005.

[133] Kaufman L. “Data Security in the World of Cloud Computing”, IEEE Security and Privacy, 7 (4), pp. 61-64, 2009.

[134] Kearney K., Torreli F. and Kotsokalis C. “SLA*: An Abstract Syntax for Service Level Agreements”, In Proc. Of 10th IEEE/ACM International Conference on Grid Computing, pp. 217-224, 2010.

[135] Keum C., Kang S., Ko I.Y., Baik J., Choi Y.I. Generating test cases for web services using extended finite state machine. In: Proc. of the 18th IFIP International Conference on Testing Communicating Systems (TestCom 2006). New York, NY, USA, May 2006.

[136] Khan K.M. and Malluhi Q. "Establishing Trust in Cloud Computing," IT Professional , 12 (5), pp.20-27, Sept.-Oct. 2010.

[137] King T.M. and Ganti A.S. “Migrating autonomic self-testing to the cloud”. In Proceedings of ICSTW 2010, 2010.

[138] Kiran M., Jiang M., Armstrong D., Djemame K. “Towards a Service Lifecycle Based Methodology for Risk Assessment in Cloud Computing”, In Proc. of the International Conference on Cloud and Green Computing (CGC'2011), December 2011.

[139] Kloti R., Kotronis V. and Smith P. “OpenFlow: A Security Analysis”. In 8th Workshop on Secure Network Protocols (NPSec 2013), Germany, October 2013.

[140] Kloukinas C., Spanoudakis G., and Mahbub K. “Estimating Event Lifetimes for Distributed Runtime Verification”, In Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, July 2008.

[141] Kramler G., Kapsammer E., Kappel G. and Retschitzegger W. “Towards using uml 2 for modeling web service collaboration protocols”. In Proceedings of the first international conference on interoperability of enterprise software and applications (INTEROP–ESA05). Springer, 2005.

[142] Krautheim F.J. “Private virtual infrastructure for cloud computing”. In Proceedings of HotCloud 2009, San Diego, CA, USA, 2009.

[143] Krotsiani M. and Spanoudakis G. “Continuous Certification of Non-Repudiation in Cloud Storage Services”, In Proceedings of the 4th IEEE International Symposium on Trust and Security in Cloud Computing (IEEE TSCloud 2014), Beijing, China, September 2014.

[144] Krotsiani M., Spanoudakis G. and Kloukinas C. “Monitoring-Based Certification of Cloud Service Security”, In proceedings of the On the Move to Meaningful Internet Systems: OTM 2015 Conferences, Vol. 9415 of the series Lecture Notes in Computer Science, pp. 644-659, October 2015.

[145] Krotsiani M., Spanoudakis G., Mahbub K. “Incremental Certification of Cloud Services”. In Proc. of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013), Barcelona, pp. 72-80, 2013.

[146] Kwiatkowska M., Norman G. and Parker D. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In Proc. of the 23rd International Conference on Computer Aided Verification (CAV’11), volume 6806 of LNCS, pp. 585-591, Springer, 2011.

[147] Kwiatkowska M., Norman G. and Parker D. “Probabilistic Model Checking: Part 2 – Descrete – Time Markov chains” Available from: http://www.prismmodelchecker.org/lectures/biss07/02-dtmcs.pdf [retrieved March 2015]

[148] Kwiatkowska M., Norman G. and Parker D. “Probabilistic Symbolic Model Checking with {PRISM}: A Hybrid Approach”. International Journal on Software Tools for Technology Transfer (STTT), 6(2), pp. 128-142, 2004.

[149] Lagazio M., Barnard-Wills D., Rodrigues R., Wright D. “Certification Schemes for Cloud Computing”, EU Commission Report, Digital Agenta for Europe, 2014. Available from: https://ec.europa.eu/digital-agenda/en/news/certification-schemes-cloud-computing [retrieved 2015]

[150] Li H., Dai Y. and Yang B. "Identity-Based Cryptography for Cloud Security", IACR Cryptology ePrint Archive, pp.169-169, 2011.

[151] Li M., Zang W., Bai K., Yu M. and Liu P. “MyCloud: Supporting user-configured privacy protection in cloud computing”. In Proceedings of ACSAC 2013, 2013.

[152] Likitalo V. “Remote Attestation and Peer-to-Peer Networks”. In Helsinki University of Technology Laboratory of Information Processing Science. Available from: http://www.tml.tkk.fi/Publications/C/18/likitalo.pdf [retrieved 2013]

[153] Lipner S. “Twenty years of evaluation criteria and commercial technology”. In Proccedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 1999.

[154] Liu F., Tong J., Mao J., Bohn R., Messina J., Badger L., Leaf D. “NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292). CreateSpace Independent Publishing Platform, USA, 2012.

[155] Lodderstedt T., Basin D., Doser J. et al. “SecureUML: A UML–based modeling language for model–driven security”. Lecture notes in computer science, pp. 426–441, 2002.

[156] Lombardi F. and Di Pietro R. “Secure virtualization for cloud computing”, J. Netw. Comput. Appl. 34 (4), pp. 1113-1122, July 2011.

[157] Lorenzoli D. and Spanoudakis G. “Predicting Software Service Availability: Towards a Runtime Monitoring Approach”, IEEE 9th International Conference on Web Services (Work-in-Progress track), July 2011

[158] Lorenzoli D., Spanoudakis G. “Runtime Prediction of Software Service Availability”, 2011 International Conference on Software Engineering Research and Practice (SERP'11), July 18-21, pp. 239-245, 2011

[159] Ma W., Wu Q. and Tan Y. “A TPA Based Efficient Non-Repudiation Scheme for Cloud Storage”. In Proceedings of the Second International Conference on Systems Engineering and Modeling (ICSEM -13), pp. 1630-1635, 2013.

[160] Mahbub K. and Spanoudakis G. "Monitoring WS-Agreements: An Event Calculus Based Approach" Springer monograph on Test and Analysis of Web Services, (Eds) Baresi L. and Di Nitto E., Springer Verlang, pp. 265-306, 2007.

[161] Mahbub K. and Spanoudakis G. "A framework for Requirements Monitoring of Service Based Systems", 2nd International Conference on Service Oriented Computing (ICSOC 2004), pp 84 – 93, November 2004.

[162] Mahbub K., Spanoudakis G. and Tsigkritis T. "Translation of SLAs into Monitoring Specifications", In Service Level Agreements for Cloud Computing, R. Yahyapour, P. Weider (Eds), Springer-verlag, 2011.

[163] Mana A., Koshutanski H. and Pérez E.J., “A trust negotiation based security framework for service provisioning in load-balancing clusters”. Computers & Security 31(1), pp. 4-25, 2012.

[164] Mao C. Towards a hierarchical testing and evaluation strategy for web services system. In: Proc. of the 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2009). Haikou, China, December 2009.

[165] Markowitch O. and Roggeman Y. “Probabilistic Non-Repudiation without trusted third party”. In Proceedings of the Second Conference on Security in Communication Networks, 1999.

[166] Masayuki O., Shiozaki T. and Suzuki T. "Security architecture for cloud computing." Fujitsu scientific and technical journal, 46(4), pp. 397-402, 2010.

[167] Massie M.L., Chun B.N. and Culler D.E. “The ganglia distributed monitoring system: design, implementation, and experience”, Parallel Computing, vol. 30, 2004, pp. 817–840.

[168] Mayrhofer R., Radi H., Ferscha A. “Recognizing and Predicting Context by Learning from User Behavior”. Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia, 1(1), May 2004

[169] McAfee Cloud Service, Available from: http://www.mcafee.com/uk/resources/data-sheets/ds-cloud-secure-program.pdf [retrieved 2014]

[170] McAfee MySQL AUDIT Plugin, Available From: https://github.com/mcafee/mysql-audit

[171] McAndrew T. “FISMA vs. FedRAMP: A Coalfire Perspective”, Dallas, April 2012. Available from: http://www.coalfire.com/medialib/assets/PDFs/Perspectives/Coalfire-Perspective-FISMA-vs-FedRAMP.pdf

[172] Meinxner F. and Buettner R. “Trust as an Integral Part for Success of Cloud Computing”. Seventh International Conference on Internet and Web Applications and Services (ICIW 2012), pp. 207 - 214, 2012.

[173] Mell P. and Grance T. “Effectively and Securely Using the Cloud Computing Paradigm (v.025)”, presentation, US National Isntitute of Standards and Technology (NIST), 2009.

[174] Microsoft, The Economics of Cloud for the EU Public Sector, Paper, 2010, Avalibe from: http://www.microsoft.eu/Portals/0/Document/EU_Public_Sector_Cloud_Economics_A4.pdf [retrieved: June, 2013].

[175] Molnar D. and Schechter S.E. “Self Hosting vs. Cloud Hosting: Accounting for the Security Impact of Hosting in the Cloud”. In Workshop on the Economics of Information Security (WEIS)], Cambridge, MA, USA, June 2010.

[176] Morris T. “Trusted platform module. In Encyclopedia of Cryptography and Security”, (eds) H.C.A. van Tilborg and S. Jajodia, Springer, 2011.

[177] Moser O., Rosenberg F. and Dustdar S. “Non-intrusive monitoring and service adaptation for WS-BPEL” In Proceedings of the 17th international conference on World Wide Web (WWW '08). ACM, New York, NY, USA, pp. 815-824, 2008.

[178] Munoz A. and Mana A. “Bridging the GAP between software certification and trusted computing for securing cloud computing”. In Proceedings of the IEEE SERVICES 2013, June 2013.

[179] Nagios, 2011. http://www.nagios.org/ (last accessed on 11/10/2013)

[180] NIST 800-53 rev. 3, Available from: http://www.nist.org/nist_plugins/content/content.php?content.18

[181] Novikoff E. "The role of remote monitoring in Cloud Computing", http://www.enkiconsulting.net/blog/the-role-of-remote-monitoring-in-cloud-computing.html. (accessed on 17/10/2013).

[182] Open Certification Framework, https://cloudsecurityalliance.org/research/ocf/

[183] Ovum, http://www.computing.co.uk/ctg/news/2113323/ovum-public-cloud-services-market-explode

[184] Paquette S., Jaeger P.T., Wilson S.C. “Identifying the security risks associated with governmental use of cloud computing”, Government Information Quarterly, 27(3), pp. 245-253, 2010.

[185] Parker D. “Implementation of Symbolic Model Checking for Probabilistic Systems”, 2002. Available from: http://163.1.88.73/papers/davesthesis.pdf [retrieved March 2015]

[186] PCI Security Standards Council, PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard v2, https://www.pcisecuritystandards.org/documents/PCI%20SSC%20Quick%20Reference%20Guide.pdf [retrieved: June, 2013]

[187] Pino L. and Spanoudakis G. “Constructing Secure Service Compositions with patterns” 2012 IEEE 8th World Congress on Services, pp. 184-191, 2012.

[188] Pino L. and Spanoudakis G. “Finding Secure Compositions of Software Services: Towards A Pattern Based Approach”. In 5th IFIP Int. Conf. on New Technologies, Mobility and Security – Track on Security, Istanbul, Turkey, May 2012.

[189] Pino L., Mahbub K. and Spanoudakis G. “Designing Secure Service Workflows in BPEL”. In Porceedings of the 12th International Conference on Service Oriented Computing (ICSOC 2014), Paris, November 2014.

[190] Pino L., Spanoudakis G., Fuchs A., Gurgens S. “Discovering Secure Service Compositions”. In Proceedings of the 4th International Conference on Cloud Computing and Services Sciences, Barcelona, Spain, April 2014.

[191] Prism Model Checker, http://www.prismmodelchecker.org/

[192] Rajendran T., Balasubramanie P., Cherian E. “An efficient WS-QoS broker based architecturefor web services selection”. International Journal of Computer Applications 1(9), pp. 79–84, 2010.

[193] Ramgovind S., Eloff M.M., Smith E. “The Management of Security in Cloud Computing”, Information Security for South Africa (ISSA), pp. 1–7, 2010.

[194] Ran S. “A model for web services discovery with QoS”. ACM SIGecom Exchanges 4, 1, 1–10, 2003.

[195] Ratha N.K., Connell J.H. and Bolle R.M. "Enhancing security and privacy in biometrics-based authentication systems." IBM systems Journal 40(3), pp. 614-634, 2001.

[196] RUBiS Benchmark, Available Form: http://rubis.ow2.org/

[197] Rumbaugh J., Jacobson I. and Booch G. “Unified Modeling Language Reference Manual”. The Pearson Higher Education, 2004.

[198] Sabahi F. "Secure Virtualization for Cloud Environment Using Hypervisor-based Technology", International Journal of Machine Learning and Computing, 2(1), pp. 39-45, February 2012.

[199] Salifu M. et al. Using Problem Descriptions to Represent Variability for Context-Aware Application. in 1st VaMoS 07. 2007. Limerick, Ireland: Lero Technical Report.

[200] Salifu M., Nuseibeh Y.Y.B. “Specifying Monitoring and Switching Problems in Context”. In Proccedings of the 15th IEEE International Requirements Engineering Conference (RE ’07), pp. 211–220, 2007.

[201] Salva S. and Rabhi I. “Automatic web service robustness testing from wsdl descriptions”. In: Proccedings of the 12th European Workshop on Dependable Computing (EWDC 2009). Toulouse, France, May 2009

[202] Sangroya A., Kumar S., Dhok J. and Varma V. “Towards Analysing Data Security Risks in Cloud Computing Environments”, In Proceedings of the 4th International Conference of Information Systems, Technology and Management (ICISTM2010), pp. 255-265, Thailand, March 2010.

[203] Santos N., Rodrigues R., Gummadi K.P. and Saroiu S. “Policy-sealed data: A new abstraction for building trusted cloud services”. In Proceedings of USENIX Security Symposium, 2012.

[204] Sarathy V., Narayan P. and Mikkilineni R. “Next generation Cloud Computing Architecture. Enabling real-time dynamism for shared distributed physical infrastructure”, In Proceedings of the 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE '10), 2010.

[205] Saripalli P. and Walters B. “QUIRC: A Quantitative Impact and Risk Assessment Framework For Cloud Security”. In Proc. Of the 3rd IEEE International Conference on Cloud Computing, IEEE, pp. 280 – 288, 2010.

[206] Schubert L. “The Future of Cloud Computing Opportunities for European Cloud Computing Beyond 2010”, [online] Available from: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf (Last accessed on 07/07/2012).

[207] Securing Web services for army SOA, http://www.sei.cmu.edu/solutions/softwaredev/securing-web-services.cfm

[208] SERENITY Project. Serenity, system engineering for security & dependability. www.serenity–project.org, 2006.

[209] Serhani M., Dssouli R., Hafid A., Sahraoui H. “A QoS broker based architecture for efficient web services selection”. In Proc. of the IEEE International Conference on Web Services (ICWS 2005). Orlando, FL, USA, July 2005.

[210] Shanahan M. “The event calculus explained”, In Artificial Intelligence Today [Ed] Wooldridge, M. J. and Veloso, M., Vol. 1600, LNCS, Springer, pp. 409-430, 1999.

[211] Shar L.K. and Kuan H. B. “Defending against Cross-Site Scripting Attacks”. Computer, 45(3), pp. 55–62, 2012

[212] Sheldon F.T., Weber J. M., Yoo S M. and Pan W.D. “The Insecurity of Wireless Networks”. IEEE Security & Privacy, 10(4), pp. 54–61, 2012.

[213] Shi W., Lee J.H., Suh T., Woo D.H., Zhang X. “Architectural support of multiple hypervisors over single platform for enhancing cloud computing security”. In Procceefings of the ninth Conference of Computing Frontiers (CF ’12), pp.75-84, 2012.

[214] Simmonds J., Gan Y., Chechik M., Nejati S., O'Farrell B., Litani E., Waterhouse J. "Runtime Monitoring of Web Service Conversations," IEEE Transactions on Services Computing. IEEE computer Society Digital Library. IEEE Computer Society, pp. 223-244, 2009.

[215] Skogan D., Groenmo R., Solheim I., Inf S., Technol C., and Oslo N. “Web service composition in UML”. In Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings, pages 47–57, 2004.

[216] Spanoudakis G. and Mahbub K. “Non intrusive monitoring of service based systems”. Internatinal Journals of Cooperative Information Systems, 15(3), pp. 325–358, 2006.

[217] Spanoudakis G., Damiani E. and Mana A. “Certifying services in cloud: The case for a hybrid, incremental and multi-layer approach”. In Proceedings of the IEEE HASE 2012, October 2012.

[218] Spanoudakis G., Kloukinas C. and Mahbub K. “The SERENITY Runtime Monitoring Framework, In Security and Dependability for Ambient Intelligence”, In Security and Dependability for Ambient Intelligence, (Eds) Spanoudakis G., Mana A., Kokolakis S. “Advances in Information Security Series”, Springer, ISBN-978-0-387-88775-3, pp. 213-238, 2009.

[219] Spring, J., Monitoring Cloud Computing by Layer, Part 1, IEEE Security and Privacy 9(2), pp. 66-68, 2011.

[220] Spring, J., Monitoring Cloud Computing by Layer, Part 2, IEEE Security and Privacy, 9(3), pp. 52-55, 2011.

[221] Sugerman J., Venkitachalam G. and Lim B.H. "Virtualizing I/O devices on VMware workstation’s hosted virtual machine monitor." In Procceedings of the General Track: 2002 USENIX Annual Technical Conference, (Ed.) Y. Park, pp. 1-14, 2002.

[222] Szefer E.J., Keller E., Lee R.B., Rexford J. "Eliminating the Hypervisor Attack Surface for a More Secure Cloud", In Procceedings of the 18th ACM Conference on Computer and Communications Security, pp. 401-412, 2011.

[223] Szefer J. and Lee R.B. "Architectural Support for Hypervisor-Secure Virtualization". In Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, pp. 437-450, 2012.

[224] Takabi H., Joshi J.B.D. and Ahn G.J. "Security and privacy challenges in cloud computing environments." IEEE Security & Privacy 6(8), pp. 24-31, 2010.

[225] The 2011 Mid-Year Top Cyber Security Risks Report, Technical White Paper, HP Enterprise, 2011, [online] Available at: http://www.hpenterprisesecurity.com/collateral/report/2011FullYearCyberSecurityRisksReport.pdf [retrieved June 2012]

[226] Truong H.L. and Fahringer T. "SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid," Scientific Programming, 12(4), pp. 225–237, November 2004, axGrids 2004 Special Issue.

[227] TrustCOM, 2007. The TrustCOM project. Deliverable 64: Final TrustCoM Reference implementation and associated tools and user manual. June 2007 (v3.0).

[228] Truste, Available at http://www.truste.com/

[229] Trusted Computing Group: TCG Specifications. Available from: https://www.trustedcomputinggroup.org/specs/ [retrieved 2013]

[230] Tsai W., Paul R., Yamin W., Chun F., Dong W.L. "Extending WSDL to facilitate web services testing". In Proc. of the 7th IEEE International Symposium on High Assurance Systems Engineering. Tokyo, Japan, October 2002.

[231] Tsigritis T., Spanoudakis G. "Diagnosing Runtime Violations of Security & Dependability Properties", In Proccedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 661-666, July 2008.

[232] Tsigritis T., Spanoudakis G., Kloukinas C., Lorenzoli D. "Diagnosis and Threat detection capabilities of the SERENITY Runtime Framework", In Security and Dependability for Ambient Intelligence, (eds) G. Spanoudakis, A. Mana, Kokolakis, Advances in Information Security Series, Springer, pp 239-272, 2009.

[233] Vaclav M.J. "Biometric authentication systems", 2000. [online] available from: http://www.fi.muni.cz/reports/files/older/FIMU-RS-2000-08.pdf [retrieved August 2015].

[234] Yeo C.S., Buyya R., Integrated Risk Analysis for a Commercial Computing Service, Journal of Grid Computing, 7(1), pp. 1-24, 2009.

[235] Zech P. "Risk-based security testing in cloud computing environments". In Proceedings of IEEE ICST 2011, 2011.

[236] Zhou J. and Gollmann D. "A Fair Non-Repudiation Protocol" In Proceedings of the IEEE Symposium on Security and Privacy, pp. 55-61, 1996.

[237] Zhou J. and Gollmann D. "An Efficient Non-Repudiation Protocol", In Proceedings of the 10th Computer Security Foundations Workshop, pp. 126-132, 1997.

GLOSSARY

Anomaly	Part of the certification model, used to analyse the behaviour of different external actors that interact with ToC that may lead to potential future attacks.
Assertion	The formal specification of a security property that is expressed in a way to allow its automated assessment.
Asset	Information, resources or applications that need to be secured.
Assurance	A positive declaration that aims to provide confidence to users.
Attack	An attempt by malicious users to damage or destroy a system or a network.
Certificate	The outcome of the certification process, which states the cloud providers' compliance, according to an assessment defined in a certification model.
Certification	A process that allows users to formulate a trusted judgment for cloud providers, according to their compliance with certain policies or standards.
Certification Model (CM)	An XML based model that defines the way to process the certification of a service for a specific security property.
Conflict	Defines an assessment of the security property for a sub period of the time specified in the certification model, which gives a different result from the assessment of the same property according to the assertion specified in the model.
Life Cycle	A state machine diagram, used to define all possible states a certificate can take during its life
Risk	The probability of occurrence of an unwanted event and its negative consequences for the stakeholders.

Security Mechanism	A security mechanism is a method, tool, or procedure for enforcing security in a system. Also knows as countermeasures.
Security Property	The security requirement imposed on a cloud service, derived from applicable laws, policies, standards or regulations, that need to be addressed and its measurements (e.g. sample size, period), in order to obtain the necessary security level.
Security Requirements	The functional and non-functional requirements that need to be satisfied in order to achieve a defined security level of a cloud service.
Target of certification (ToC*)*	An asset of a cloud service (e.g. a specific service operation, a set of service operations, data managed by the service) or an asset that is required or contributes to the realization of a cloud service (e.g., a virtual machine), which becomes the subject of certification.
Threat	Any possible event that can have a harmful impact to organizational operations, organizational assets, or individuals, through an information system via unauthorized access, destruction, disclosure, modification of information, or denial of service.

APPENDIX A: CERTIFICATION MODEL FOR NR

The CM for the NR security property is presented below, as presented in Section 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
  <p:CertificationModel xmlns:p="http://www.cumulus.org/certificate/model"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cumulus.org/certificate/model
file:/Users/Maria_K/Dropbox/Deliverables/schemas/CertificationModel_XMLSchema_v8.xsd">
    <Model_Id>cm:id:nomitoring:0001</Model_Id>
    <Signature>
      <Name>City</Name>
      <Role>CA</Role>
    </Signature>
    <TOC id="id1001">
      <providesInterface>
        <ID>001</ID>
          <ProviderRef>provider001</ProviderRef>
          <Interface>
            <InterfaceSpec>
              <Name>cloudinterface</Name>
              <Operation>
                <interfaceId>c</interfaceId>
                <OperationId>id0001</OperationId>
                <operationName>rqsac</operationName>
                <inputVariable forMatching="false" persistent="true">
                  <varName>data</varName>
                  <varType>url</varType>
                </inputVariable>
                <inputVariable forMatching="false" persistent="true">
                  <varName>t</varName>
                  <varType>url</varType>
                </inputVariable>
              </Operation>
              <Operation>
                <interfaceId>c</interfaceId>
                <OperationId>id0002</OperationId>
                <operationName>rqsbc</operationName>
                <inputVariable forMatching="false" persistent="true">
                  <varName>data</varName>
                  <varType>url</varType>
                </inputVariable>
              </Operation>
              <Operation>
                <interfaceId>c</interfaceId>
                <OperationId>id0003</OperationId>
                <operationName>rqstc</operationName>
                <inputVariable forMatching="false" persistent="true">
                  <varName>data</varName>
                  <varType>url</varType>
                </inputVariable>
              </Operation>
            </InterfaceSpec>
```

```
      </Interface>
    </providesInterface>
  </TOC>
  <SecurityProperty SecurityPropertyId="0001"
      SecurityPropertyDefinition="AIS:non-repudiation:non-repudiation-of-origin"
      Vocabulary="CSA" ShortName="AIS:non-repudiation:non-repudiation-of-origin">
    <sProperty class="http://www.cumulus-project.eu">
      <propertyPerformance>
        <propertyPerformanceRow>
          <propertyPerformanceCell name="verified">true </propertyPerformanceCell>
        </propertyPerformanceRow>
      </propertyPerformance>
    </sProperty>
    <Assertion ID="AS001">
      <InterfaceDeclr>
        <ID>001</ID>
        <ProviderRef>provider001</ProviderRef>
        <Interface>
          <InterfaceSpec>
            <Name>cloudinterface</Name>
            <Operation>
              <interfaceId>c</interfaceId>
              <OperationId>id0004</OperationId>
              <operationName>rqsac</operationName>
              <inputVariable forMatching="false" persistent="true">
                <varName>data</varName>
                <varType>url</varType>
              </inputVariable>
              <inputVariable>
                <varName>seq</varName>
                <varType>string</varType>
              </inputVariable>
              <inputVariable forMatching="false" persistent="true">
                <varName>t</varName>
                <varType>url</varType>
              </inputVariable>
            </Operation>
            <Operation>
              <interfaceId>c</interfaceId>
              <OperationId>id0005</OperationId>
              <operationName>rqsbc</operationName>
              <inputVariable forMatching="false" persistent="true">
                <varName>data</varName>
                <varType>url</varType>
              </inputVariable>
              <inputVariable>
                <varName>seq</varName>
                <varType>string</varType>
              </inputVariable>
            </Operation>
            <Operation>
              <interfaceId>c</interfaceId>
              <OperationId>id0007</OperationId>
              <operationName>rqstc</operationName>
              <inputVariable forMatching="false" persistent="true">
                <varName>data</varName>
```

```
            <varType>url</varType>
          </inputVariable>
          <inputVariable>
            <varName>seq</varName>
            <varType>string</varType>
          </inputVariable>
        </Operation>
      </InterfaceSpec>
    </Interface>
  </InterfaceDeclr>
  <VariableDeclr>
    <varName>seqrq</varName>
    <varType>string</varType>
  </VariableDeclr>
  <VariableDeclr>
    <varName>seqrs</varName>
    <varType>string</varType>
  </VariableDeclr>
  <Guaranteed ID="gt1" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <timeVariable>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
      </timeVariable>
    </quantification>
    <quantification>
      <quantifier>existential</quantifier>
      <timeVariable>
        <varName>t2</varName>
        <varType>TimeVariable</varType>
      </timeVariable>
    </quantification>
    <precondition>
      <atomicCondition conditionID="gt1-ac1">
        <eventCondition unconstrained="true">
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID0</varName>
            </eventID>
            <call>
              <interfaceId>CP</interfaceId>
              <OperationId>1</OperationId>
              <operationName>upload</operationName>
              <inputVariable forMatching="true" persistent="false">
                <varName>status1</varName>
                <varType>OpStatus</varType>
                <value>REQ-B</value>
              </inputVariable>
              <inputVariable forMatching="true" persistent="false">
                <varName>sender1</varName>
                <varType>Entity</varType>
              </inputVariable>
              <inputVariable forMatching="true" persistent="false">
                <varName>receiver1</varName>
                <varType>Entity</varType>
```

```
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>source1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>serviceId</varName>
                      <varType>string</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </inputVariable>
                  </call>
                  <tVar>
                    <timeVar>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </tVar>
                  <fromTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                  </fromTime>
                  <toTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                  </toTime>
                </event>
              </eventCondition>
            </atomicCondition>
            <WrappedCondition>
              <operator>and</operator>
              <assertionCondition>
                <atomicCondition conditionID="gt1-ac2">
                  <stateCondition>
                    <holdsAt negated="false">
                      <state name="ResUpReq">
                        <argument>
                          <variable persistent="false" forMatching="true">
                            <varName>seqrq</varName>
                            <varType>int</varType>
                          </variable>
                        </argument>
                        <argument>
                          <variable forMatching="true" persistent="false">
                            <varName>seqrs</varName>
                            <varType>int</varType>
                          </variable>
                        </argument>
                      </state>
                      <timeVar>
```

```
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </holdsAt>
              </stateCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </precondition>
      <postcondition>
        <atomicCondition conditionID="gt1-ac3">
          <eventCondition unconstrained="true">
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID1</varName>
              </eventID>
              <reply>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>upload</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>status2</varName>
                  <varType>OpStatus</varType>
                  <value>RES-B</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>sender1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>source1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>serviceId</varName>
                  <varType>string</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>seqrs</varName>
                  <varType>int</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </outputVariable>
              </reply>
              <tVar>
                <timeVar>
                  <varName>t2</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </tVar>
```

```
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                  <Expression>
                    <plus>100</plus>
                  </Expression>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
        </postcondition>
      </Guaranteed>
      <Guaranteed ID="gt2" type="Future_Formula">
        <quantification>
          <quantifier>forall</quantifier>
          <timeVariable>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVariable>
        </quantification>
        <quantification>
          <quantifier>existential</quantifier>
          <timeVariable>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVariable>
        </quantification>
        <precondition>
          <atomicCondition conditionID="gt2-ac1">
            <eventCondition unconstrained="true">
              <event>
                <eventID forMatching="true" persistent="false">
                  <varName>VID0</varName>
                </eventID>
                <call>
                  <interfaceId>CP</interfaceId>
                  <OperationId>1</OperationId>
                  <operationName>upload</operationName>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>status1</varName>
                    <varType>OpStatus</varType>
                    <value>REQ-B</value>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>sender1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
```

```
                    <varName>receiver1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>source1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </inputVariable>
                </call>
                <tVar>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
          <WrappedCondition>
            <operator>and</operator>
            <assertionCondition>
              <atomicCondition conditionID="gt2-ac2">
                <stateCondition>
                  <holdsAt negated="false">
                    <state name="ResUpReq">
                      <argument>
                        <variable persistent="false" forMatching="true">
                          <varName>seqrq</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
                      <argument>
                        <variable forMatching="true" persistent="false">
                          <varName>seqrs</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
```

```
                    </state>
                    <timeVar>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </holdsAt>
                </stateCondition>
              </atomicCondition>
            </assertionCondition>
          </WrappedCondition>
          <WrappedCondition>
            <operator>and</operator>
            <assertionCondition>
              <atomicCondition conditionID="gt2-ac3">
                <eventCondition unconstrained="false">
                  <event>
                    <eventID forMatching="true" persistent="false">
                      <varName>VID1</varName>
                    </eventID>
                    <reply>
                      <interfaceId>CP</interfaceId>
                      <OperationId>2</OperationId>
                      <operationName>upload</operationName>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>status2</varName>
                        <varType>OpStatus</varType>
                        <value>RES-B</value>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>receiver1</varName>
                        <varType>Entity</varType>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>sender1</varName>
                        <varType>Entity</varType>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>source1</varName>
                        <varType>Entity</varType>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>serviceId</varName>
                        <varType>string</varType>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>seqrs</varName>
                        <varType>int</varType>
                      </outputVariable>
                      <outputVariable forMatching="true" persistent="false">
                        <varName>seqrq</varName>
                        <varType>int</varType>
                      </outputVariable>
                    </reply>
                    <tVar>
                      <timeVar>
                        <varName>t2</varName>
```

```
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                  <Expression>
                    <plus>100</plus>
                  </Expression>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="gt2-ac4">
        <stateCondition>
          <initiates>
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID1</varName>
              </eventID>
              <reply>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>upload</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>status2</varName>
                  <varType>OpStatus</varType>
                  <value>RES-B</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>sender1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>source1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>serviceId</varName>
                  <varType>string</varType>
```

```
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </outputVariable>
                  </reply>
                  <tVar>
                    <timeVar>
                      <varName>t2</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </tVar>
                  <fromTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                  </fromTime>
                  <toTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                    <Expression>
                      <plus>100</plus>
                    </Expression>
                  </toTime>
                </event>
                <state name="ResUpReq">
                  <argument>
                    <variable persistent="false" forMatching="true">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t2</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </initiates>
            </stateCondition>
          </atomicCondition>
        </postcondition>
      </Guaranteed>
      <Guaranteed ID="gt3" type="Future_Formula">
```

```
<quantification>
  <quantifier>forall</quantifier>
  <timeVariable>
    <varName>t1</varName>
    <varType>TimeVariable</varType>
  </timeVariable>
</quantification>
<quantification>
  <quantifier>existential</quantifier>
  <timeVariable>
    <varName>t2</varName>
    <varType>TimeVariable</varType>
  </timeVariable>
</quantification>
<precondition>
  <atomicCondition conditionID="gt3-ac1">
    <eventCondition unconstrained="true">
      <event>
        <eventID forMatching="true" persistent="false">
          <varName>VID0</varName>
        </eventID>
        <call>
          <interfaceId>CP</interfaceId>
          <OperationId>1</OperationId>
          <operationName>download</operationName>
          <inputVariable forMatching="true" persistent="false">
            <varName>status1</varName>
            <varType>OpStatus</varType>
            <value>REQ-B</value>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </inputVariable>
        </call>
        <tVar>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
```

```
            </tVar>
            <fromTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </fromTime>
            <toTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </toTime>
          </event>
        </eventCondition>
      </atomicCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="gt3-ac2">
            <stateCondition>
              <holdsAt negated="false">
                <state name="ResDownReq">
                  <argument>
                    <variable persistent="false" forMatching="true">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </holdsAt>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="gt3-ac3">
        <eventCondition unconstrained="true">
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID1</varName>
            </eventID>
            <reply>
              <interfaceId>CP</interfaceId>
              <OperationId>2</OperationId>
```

```
                    <operationName>download</operationName>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>status2</varName>
                      <varType>OpStatus</varType>
                      <value>RES-B</value>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>receiver1</varName>
                      <varType>Entity</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>sender1</varName>
                      <varType>Entity</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>source1</varName>
                      <varType>Entity</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>serviceId</varName>
                      <varType>string</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </outputVariable>
                    <outputVariable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </outputVariable>
                  </reply>
                  <tVar>
                    <timeVar>
                      <varName>t2</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </tVar>
                  <fromTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                  </fromTime>
                  <toTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                    <Expression>
                      <plus>100</plus>
                    </Expression>
                  </toTime>
                </event>
              </eventCondition>
            </atomicCondition>
          </postcondition>
```

```
        </Guaranteed>
        <Guaranteed ID="gt4" type="Future_Formula">
          <quantification>
            <quantifier>forall</quantifier>
            <timeVariable>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVariable>
          </quantification>
          <quantification>
            <quantifier>existential</quantifier>
            <timeVariable>
              <varName>t2</varName>
              <varType>TimeVariable</varType>
            </timeVariable>
          </quantification>
          <precondition>
            <atomicCondition conditionID="gt4-ac1">
              <eventCondition unconstrained="true">
                <event>
                  <eventID forMatching="true" persistent="false">
                    <varName>VID0</varName>
                  </eventID>
                  <call>
                    <interfaceId>CP</interfaceId>
                    <OperationId>1</OperationId>
                    <operationName>download</operationName>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>status1</varName>
                      <varType>OpStatus</varType>
                      <value>REQ-B</value>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>sender1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>receiver1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>source1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>serviceId</varName>
                      <varType>string</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </inputVariable>
                  </call>
                  <tVar>
                    <timeVar>
                      <varName>t1</varName>
```

```
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
          <WrappedCondition>
            <operator>and</operator>
            <assertionCondition>
              <atomicCondition conditionID="gt4-ac2">
                <stateCondition>
                  <holdsAt negated="false">
                    <state name="ResDownReq">
                      <argument>
                        <variable persistent="false" forMatching="true">
                          <varName>seqrq</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
                      <argument>
                        <variable forMatching="true" persistent="false">
                          <varName>seqrs</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
                    </state>
                    <timeVar>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </holdsAt>
                </stateCondition>
              </atomicCondition>
            </assertionCondition>
          </WrappedCondition>
          <WrappedCondition>
            <operator>and</operator>
            <assertionCondition>
              <atomicCondition conditionID="gt4-ac3">
                <eventCondition unconstrained="false">
                  <event>
                    <eventID forMatching="true" persistent="false">
                      <varName>VID1</varName>
                    </eventID>
```

```
        <reply>
          <interfaceId>CP</interfaceId>
          <OperationId>2</OperationId>
          <operationName>download</operationName>
          <outputVariable forMatching="true" persistent="false">
            <varName>status2</varName>
            <varType>OpStatus</varType>
            <value>RES-B</value>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>seqrs</varName>
            <varType>int</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </outputVariable>
        </reply>
        <tVar>
          <timeVar>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
          <Expression>
            <plus>100</plus>
          </Expression>
        </toTime>
      </event>
```

```
            </eventCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="gt4-ac4">
        <stateCondition>
          <initiates>
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID1</varName>
              </eventID>
              <reply>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>download</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>status2</varName>
                  <varType>OpStatus</varType>
                  <value>RES-B</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>sender1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>source1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>serviceId</varName>
                  <varType>string</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>seqrs</varName>
                  <varType>int</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </outputVariable>
              </reply>
              <tVar>
                <timeVar>
                  <varName>t2</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </tVar>
              <fromTime>
                <time>
                  <varName>t1</varName>
```

```
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                  <Expression>
                    <plus>100</plus>
                  </Expression>
                </toTime>
              </event>
              <state name="ResDownReq">
                <argument>
                  <variable persistent="false" forMatching="true">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </variable>
                </argument>
                <argument>
                  <variable forMatching="true" persistent="false">
                    <varName>seqrs</varName>
                    <varType>int</varType>
                  </variable>
                </argument>
              </state>
              <timeVar>
                <varName>t2</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </initiates>
          </stateCondition>
        </atomicCondition>
      </postcondition>
    </Guaranteed>
    <Guaranteed ID="gt5" type="Future_Formula">
      <quantification>
        <quantifier>forall</quantifier>
        <timeVariable>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVariable>
      </quantification>
      <quantification>
        <quantifier>existential</quantifier>
        <timeVariable>
          <varName>t2</varName>
          <varType>TimeVariable</varType>
        </timeVariable>
      </quantification>
      <precondition>
        <atomicCondition conditionID="gt5-ac1">
          <eventCondition unconstrained="true">
            <event>
              <eventID forMatching="true" persistent="false">
```

```
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>CP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>resolution</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
              <value>REQ-B</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>seqrq</varName>
              <varType>int</varType>
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </fromTime>
          <toTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
    <WrappedCondition>
      <operator>and</operator>
      <assertionCondition>
```

```
            <atomicCondition conditionID="gt5-ac2">
              <stateCondition>
                <holdsAt negated="false">
                  <state name="ResResReq">
                    <argument>
                      <variable persistent="false" forMatching="true">
                        <varName>seqrq</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>seqrs</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </holdsAt>
              </stateCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </precondition>
      <postcondition>
        <atomicCondition conditionID="gt5-ac3">
          <eventCondition unconstrained="true">
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID1</varName>
              </eventID>
              <reply>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>resolution</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>status2</varName>
                  <varType>OpStatus</varType>
                  <value>RES-B</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>sender1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>source1</varName>
                  <varType>Entity</varType>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
```

```
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>seqrs</varName>
                    <varType>int</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </outputVariable>
                </reply>
                <tVar>
                  <timeVar>
                    <varName>t2</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                  <Expression>
                    <plus>100</plus>
                  </Expression>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
        </postcondition>
      </Guaranteed>
      <Guaranteed ID="gt6" type="Future_Formula">
        <quantification>
          <quantifier>forall</quantifier>
          <timeVariable>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVariable>
        </quantification>
        <quantification>
          <quantifier>existential</quantifier>
          <timeVariable>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVariable>
        </quantification>
        <precondition>
          <atomicCondition conditionID="gt6-ac1">
            <eventCondition unconstrained="true">
```

```
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID0</varName>
              </eventID>
              <call>
                <interfaceId>CP</interfaceId>
                <OperationId>1</OperationId>
                <operationName>resolution</operationName>
                <inputVariable forMatching="true" persistent="false">
                  <varName>status1</varName>
                  <varType>OpStatus</varType>
                  <value>REQ-B</value>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>sender1</varName>
                  <varType>Entity</varType>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>receiver1</varName>
                  <varType>Entity</varType>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>source1</varName>
                  <varType>Entity</varType>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>serviceId</varName>
                  <varType>string</varType>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </inputVariable>
              </call>
              <tVar>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </tVar>
              <fromTime>
                <time>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </time>
              </fromTime>
              <toTime>
                <time>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </time>
              </toTime>
            </event>
          </eventCondition>
        </atomicCondition>
        <WrappedCondition>
```

```
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="gt6-ac2">
            <stateCondition>
              <holdsAt negated="false">
                <state name="ResResReq">
                  <argument>
                    <variable persistent="false" forMatching="true">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </holdsAt>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="gt6-ac3">
            <eventCondition unconstrained="false">
              <event>
                <eventID forMatching="true" persistent="false">
                  <varName>VID1</varName>
                </eventID>
                <reply>
                  <interfaceId>CP</interfaceId>
                  <OperationId>2</OperationId>
                  <operationName>resolution</operationName>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>status2</varName>
                    <varType>OpStatus</varType>
                    <value>RES-B</value>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>receiver1</varName>
                    <varType>Entity</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>sender1</varName>
                    <varType>Entity</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>source1</varName>
```

```
                    <varType>Entity</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>seqrs</varName>
                    <varType>int</varType>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </outputVariable>
                </reply>
                <tVar>
                  <timeVar>
                    <varName>t2</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                  <Expression>
                    <plus>100</plus>
                  </Expression>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="gt6-ac4">
        <stateCondition>
          <initiates>
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID0</varName>
              </eventID>
              <reply>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>resolution</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>status2</varName>
```

```
            <varType>OpStatus</varType>
            <value>RES-B</value>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>seqrs</varName>
            <varType>int</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </outputVariable>
        </reply>
        <tVar>
          <timeVar>
            <varName>t2</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
          <Expression>
            <plus>100</plus>
          </Expression>
        </toTime>
      </event>
      <state name="ResResReq">
        <argument>
          <variable persistent="false" forMatching="true">
            <varName>seqrq</varName>
            <varType>int</varType>
          </variable>
```

```
                    </argument>
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>seqrs</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t2</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </initiates>
              </stateCondition>
            </atomicCondition>
          </postcondition>
        </Guaranteed>
      </Assertion>
    </SecurityProperty>
    <AssessmentScheme>
      <EvidenceSufficiencyCondition Id="1011">
        <MonitoringPeriodCondition minMonitoredPeriod="3" periodUnit="days"/>
        <ExpectedSystemOperationModel>
          <InitialState stateId="s1" name="state1"/>
          <states>
            <state>
              <atomicState stateId="s2" name="state2"/>
            </state>
            <state>
              <atomicState stateId="s3" name="state3"/>
            </state>
            <state>
              <atomicState stateId="s4" name="state4"/>
            </state>
          </states>
          <transitions>
            <transition Id="t1" From="s1" To="s2" lPr="0.1" uPr="0.5">
              <CallEvent>
                <invocation>
                  <execute>
                    <interfaceId>CP</interfaceId>
                    <OperationId>1</OperationId>
                    <operationName>upload</operationName>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>sender</varName>
                      <varType>String</varType>
                      <value>DP</value>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>receiver</varName>
                      <varType>String</varType>
                      <value>CP</value>
                    </inputVariable>
                  </execute>
                </invocation>
              </CallEvent>
```

```
        </transition>
        <transition Id="t2" From="s2" To="s1" lPr="1" uPr="1">
          <ReplyEvent>
            <invocation>
              <execute>
                <interfaceId>CP</interfaceId>
                <OperationId>1</OperationId>
                <operationName>upload</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver</varName>
                  <varType>String</varType>
                  <value>CP</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
                  <varName>sender</varName>
                  <varType>String</varType>
                  <value>DP</value>
                </outputVariable>
              </execute>
            </invocation>
          </ReplyEvent>
        </transition>
        <transition Id="t3" From="s1" To="s3" lPR="0.0" uPr="0.2">
          <CallEvent>
            <invocation>
              <execute>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>download</operationName>
                <inputVariable forMatching="true" persistent="false">
                  <varName>sender</varName>
                  <varType>String</varType>
                  <value>DC</value>
                </inputVariable>
                <inputVariable forMatching="true" persistent="false">
                  <varName>receiver</varName>
                  <varType>String</varType>
                  <value>CP</value>
                </inputVariable>
              </execute>
            </invocation>
          </CallEvent>
        </transition>
        <transition Id="t4" From="s3" To="s1" lPr="1" uPr="1">
          <ReplyEvent>
            <invocation>
              <execute>
                <interfaceId>CP</interfaceId>
                <OperationId>2</OperationId>
                <operationName>download</operationName>
                <outputVariable forMatching="true" persistent="false">
                  <varName>receiver</varName>
                  <varType>String</varType>
                  <value>CP</value>
                </outputVariable>
                <outputVariable forMatching="true" persistent="false">
```

```
                    <varName>sender</varName>
                    <varType>String</varType>
                    <value>DC</value>
                  </outputVariable>
                </execute>
              </invocation>
            </ReplyEvent>
          </transition>
          <transition Id="t5" From="s1" To="s4" lPr="0.5" uPr="1">
            <CallEvent>
              <invocation>
                <execute>
                  <interfaceId>CP</interfaceId>
                  <OperationId>3</OperationId>
                  <operationName>resolution</operationName>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>sender</varName>
                    <varType>String</varType>
                    <value>TTP</value>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>receiver</varName>
                    <varType>String</varType>
                    <value>CP</value>
                  </inputVariable>
                </execute>
              </invocation>
            </CallEvent>
          </transition>
          <transition Id="t6" From="s4" To="s1" lPr="1" uPr="1">
            <ReplyEvent>
              <invocation>
                <execute>
                  <interfaceId>CP</interfaceId>
                  <OperationId>3</OperationId>
                  <operationName>resolution</operationName>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>receiver</varName>
                    <varType>String</varType>
                    <value>CP</value>
                  </outputVariable>
                  <outputVariable forMatching="true" persistent="false">
                    <varName>sender</varName>
                    <varType>String</varType>
                    <value>TTP</value>
                  </outputVariable>
                </execute>
              </invocation>
            </ReplyEvent>
          </transition>
        </transitions>
      </ExpectedSystemOperationModel>
    </EvidenceSufficiencyCondition>
    <ExpirationCondition Id="987">
      <elapsedPeriod period="1" periodUnit="years"/>
    </ExpirationCondition>
```

```
<Anomalies>
  <Assertion ID="AS002">
  <InterfaceDeclr>
    <ID>001</ID>
    <ProviderRef>provider001</ProviderRef>
    <Interface>
      <InterfaceSpec>
        <Name>cloudinterface</Name>
        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0004</OperationId>
          <operationName>rqsac</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
          <inputVariable>
            <varName>seq</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="false" persistent="true">
            <varName>t</varName>
            <varType>url</varType>
          </inputVariable>
        </Operation>
        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0005</OperationId>
          <operationName>rqsbc</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
          <inputVariable>
            <varName>seq</varName>
            <varType>string</varType>
          </inputVariable>
        </Operation>
        <Operation>
          <interfaceId>c</interfaceId>
          <OperationId>id0007</OperationId>
          <operationName>rqstc</operationName>
          <inputVariable forMatching="false" persistent="true">
            <varName>data</varName>
            <varType>url</varType>
          </inputVariable>
          <inputVariable>
            <varName>seq</varName>
            <varType>string</varType>
          </inputVariable>
        </Operation>
      </InterfaceSpec>
    </Interface>
  </InterfaceDeclr>
  <Guaranteed ID="gt7" type="Future_Formula">
    <quantification>
```

```
      <quantifier>forall</quantifier>
      <regularVariable>
        <varName>systime</varName>
        <varType>TimeVariable</varType>
      </regularVariable>
    </quantification>
    <postcondition>
      <atomicCondition conditionID="gt7-ac1">
        <stateCondition>
          <initially>
            <state name="RepeatedUpReq">
              <argument>
                <variable forMatching="true" persistent="false">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </variable>
              </argument>
              <argument>
                <variable forMatching="true" persistent="false">
                  <varName>N</varName>
                  <varType>int</varType>
                  <value>0</value>
                </variable>
              </argument>
            </state>
            <timeVar>
              <varName>systime</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </initially>
        </stateCondition>
      </atomicCondition>
    </postcondition>
  </Guaranteed>
  <Guaranteed ID="gt8" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <timeVariable>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
      </timeVariable>
    </quantification>
    <precondition>
      <atomicCondition conditionID="gt8-ac1">
        <eventCondition unconstrained="true">
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID0</varName>
            </eventID>
            <call>
              <interfaceId>CP</interfaceId>
              <OperationId>1</OperationId>
              <operationName>upload</operationName>
              <inputVariable forMatching="true" persistent="false">
                <varName>status1</varName>
                <varType>OpStatus</varType>
```

```
            <value>REQ-B</value>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </inputVariable>
        </call>
        <tVar>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </toTime>
      </event>
    </eventCondition>
  </atomicCondition>
  <WrappedCondition>
    <operator>and</operator>
    <assertionCondition>
      <atomicCondition conditionID="gt8-ac2">
        <stateCondition>
          <holdsAt negated="true">
            <state name="ResUpReq">
              <argument>
                <variable persistent="false" forMatching="true">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </variable>
```

```
                    </argument>
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>seqrs</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </holdsAt>
              </stateCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </precondition>
      <postcondition>
        <atomicCondition conditionID="gt8-ac3">
          <stateCondition>
            <terminates>
              <event>
                <eventID forMatching="true" persistent="false">
                  <varName>VID0</varName>
                </eventID>
                <call>
                  <interfaceId>CP</interfaceId>
                  <OperationId>1</OperationId>
                  <operationName>upload</operationName>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>status1</varName>
                    <varType>OpStatus</varType>
                    <value>REQ-B</value>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>sender1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>receiver1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>source1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </inputVariable>
                </call>
```

```
                    <tVar>
                      <timeVar>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                      </timeVar>
                    </tVar>
                    <fromTime>
                      <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                      </time>
                    </fromTime>
                    <toTime>
                      <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                      </time>
                    </toTime>
                  </event>
                  <state name="RepeatedUpReq">
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>seqrq</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>N</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </terminates>
              </stateCondition>
            </atomicCondition>
            <WrappedCondition>
              <operator>and</operator>
              <assertionCondition>
                <atomicCondition conditionID="gt8-ac4">
                  <stateCondition>
                    <initiates>
                      <event>
                        <eventID forMatching="true" persistent="false">
                          <varName>VID0</varName>
                        </eventID>
                        <call>
                          <interfaceId>CP</interfaceId>
                          <OperationId>1</OperationId>
                          <operationName>upload</operationName>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>status1</varName>
```

```
            <varType>OpStatus</varType>
            <value>REQ-B</value>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>sender1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>receiver1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>source1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </inputVariable>
        </call>
        <tVar>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </toTime>
      </event>
      <state name="RepeatedUpReq">
        <argument>
          <variable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </variable>
        </argument>
        <argument>
          <operationCall>
            <name>add</name>
            <partner>self</partner>
            <variable forMatching="true" persistent="false">
              <varName>N</varName>
```

```
                        <varType>int</varType>
                      </variable>
                      <constant>
                        <name>Counter</name>
                        <value>1</value>
                      </constant>
                    </operationCall>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </initiates>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </postcondition>
  </Guaranteed>
  <Guaranteed ID="gt9" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <regularVariable>
        <varName>systime</varName>
        <varType>TimeVariable</varType>
      </regularVariable>
    </quantification>
    <postcondition>
      <atomicCondition conditionID="gt9-ac1">
        <stateCondition>
          <initially>
            <state name="RepeatedDownReq">
              <argument>
                <variable forMatching="true" persistent="false">
                  <varName>seqrq</varName>
                  <varType>int</varType>
                </variable>
              </argument>
              <argument>
                <variable forMatching="true" persistent="false">
                  <varName>N</varName>
                  <varType>int</varType>
                  <value>0</value>
                </variable>
              </argument>
            </state>
            <timeVar>
              <varName>systime</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </initially>
        </stateCondition>
      </atomicCondition>
    </postcondition>
  </Guaranteed>
```

```
<Guaranteed ID="gt10" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="gt10-ac1">
      <eventCondition unconstrained="true">
        <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>CP</interfaceId>
            <OperationId>1</OperationId>
            <operationName>download</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
              <value>REQ-B</value>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>seqrq</varName>
              <varType>int</varType>
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
```

```
            </fromTime>
            <toTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </toTime>
          </event>
        </eventCondition>
      </atomicCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="gt10-ac2">
            <stateCondition>
              <holdsAt negated="true">
                <state name="ResDownReq">
                  <argument>
                    <variable persistent="false" forMatching="true">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>seqrs</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </holdsAt>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="gt10-ac3">
        <stateCondition>
          <terminates>
            <event>
              <eventID forMatching="true" persistent="false">
                <varName>VID0</varName>
              </eventID>
              <call>
                <interfaceId>CP</interfaceId>
                <OperationId>1</OperationId>
                <operationName>download</operationName>
                <inputVariable forMatching="true" persistent="false">
                  <varName>status1</varName>
                  <varType>OpStatus</varType>
                  <value>REQ-B</value>
```

```
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>sender1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>receiver1</varName>
          <varType>Entity</varType>
        </inputVariable>
         <inputVariable forMatching="true" persistent="false">
          <varName>source1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>serviceId</varName>
          <varType>string</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>seqrq</varName>
          <varType>int</varType>
        </inputVariable>
      </call>
      <tVar>
        <timeVar>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVar>
      </tVar>
      <fromTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
      </fromTime>
      <toTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
      </toTime>
    </event>
    <state name="RepeatedDownReq">
      <argument>
        <variable forMatching="true" persistent="false">
          <varName>seqrq</varName>
          <varType>int</varType>
        </variable>
      </argument>
      <argument>
        <variable forMatching="true" persistent="false">
          <varName>N</varName>
          <varType>int</varType>
        </variable>
      </argument>
    </state>
    <timeVar>
```

```
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </terminates>
              </stateCondition>
            </atomicCondition>
            <WrappedCondition>
              <operator>and</operator>
              <assertionCondition>
                <atomicCondition conditionID="gt10-ac4">
                  <stateCondition>
                    <initiates>
                      <event>
                        <eventID forMatching="true" persistent="false">
                          <varName>VID0</varName>
                        </eventID>
                        <call>
                          <interfaceId>CP</interfaceId>
                          <OperationId>1</OperationId>
                          <operationName>download</operationName>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>status1</varName>
                            <varType>OpStatus</varType>
                            <value>REQ-B</value>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>sender1</varName>
                            <varType>Entity</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>receiver1</varName>
                            <varType>Entity</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>source1</varName>
                            <varType>Entity</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>serviceId</varName>
                            <varType>string</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>seqrq</varName>
                            <varType>int</varType>
                          </inputVariable>
                        </call>
                        <tVar>
                          <timeVar>
                            <varName>t1</varName>
                            <varType>TimeVariable</varType>
                          </timeVar>
                        </tVar>
                        <fromTime>
                          <time>
                            <varName>t1</varName>
                            <varType>TimeVariable</varType>
```

```
                    </time>
                  </fromTime>
                  <toTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
                  </toTime>
                </event>
                <state name="RepeatedDownReq">
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <operationCall>
                      <name>add</name>
                      <partner>self</partner>
                      <variable forMatching="true" persistent="false">
                        <varName>N</varName>
                        <varType>int</varType>
                      </variable>
                      <constant>
                        <name>Counter</name>
                        <value>1</value>
                      </constant>
                    </operationCall>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </initiates>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </postcondition>
  </Guaranteed>
  <Guaranteed ID="gt11" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <regularVariable>
        <varName>systime</varName>
        <varType>TimeVariable</varType>
      </regularVariable>
    </quantification>
    <postcondition>
      <atomicCondition conditionID="gt11-ac1">
        <stateCondition>
          <initially>
            <state name="RepeatedResReq">
              <argument>
```

```
                    <variable forMatching="true" persistent="false">
                      <varName>seqrq</varName>
                      <varType>int</varType>
                    </variable>
                  </argument>
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>N</varName>
                      <varType>int</varType>
                      <value>0</value>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>systime</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </initially>
            </stateCondition>
          </atomicCondition>
        </postcondition>
      </Guaranteed>
      <Guaranteed ID="gt12" type="Future_Formula">
        <quantification>
          <quantifier>forall</quantifier>
          <timeVariable>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVariable>
        </quantification>
        <precondition>
          <atomicCondition conditionID="gt12-ac1">
            <eventCondition unconstrained="true">
              <event>
                <eventID forMatching="true" persistent="false">
                  <varName>VID0</varName>
                </eventID>
                <call>
                  <interfaceId>CP</interfaceId>
                  <OperationId>1</OperationId>
                  <operationName>resolve</operationName>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>status1</varName>
                    <varType>OpStatus</varType>
                    <value>REQ-B</value>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>sender1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>receiver1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>source1</varName>
```

```
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </inputVariable>
                </call>
                <tVar>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
                <toTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </toTime>
              </event>
            </eventCondition>
          </atomicCondition>
          <WrappedCondition>
            <operator>and</operator>
            <assertionCondition>
              <atomicCondition conditionID="g128-ac2">
                <stateCondition>
                  <holdsAt negated="true">
                    <state name="ResResReq">
                      <argument>
                        <variable persistent="false" forMatching="true">
                          <varName>seqrq</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
                      <argument>
                        <variable forMatching="true" persistent="false">
                          <varName>seqrs</varName>
                          <varType>int</varType>
                        </variable>
                      </argument>
                    </state>
                    <timeVar>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
```

```
                </holdsAt>
              </stateCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </precondition>
      <postcondition>
        <atomicCondition conditionID="gt12-ac3">
          <stateCondition>
            <terminates>
              <event>
                <eventID forMatching="true" persistent="false">
                  <varName>VID0</varName>
                </eventID>
                <call>
                  <interfaceId>CP</interfaceId>
                  <OperationId>1</OperationId>
                  <operationName>resolve</operationName>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>status1</varName>
                    <varType>OpStatus</varType>
                    <value>REQ-B</value>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>sender1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>receiver1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>source1</varName>
                    <varType>Entity</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>serviceId</varName>
                    <varType>string</varType>
                  </inputVariable>
                  <inputVariable forMatching="true" persistent="false">
                    <varName>seqrq</varName>
                    <varType>int</varType>
                  </inputVariable>
                </call>
                <tVar>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </tVar>
                <fromTime>
                  <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </time>
                </fromTime>
```

```
                    <toTime>
                      <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                      </time>
                    </toTime>
                  </event>
                  <state name="RepeatedResReq">
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>seqrq</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                    <argument>
                      <variable forMatching="true" persistent="false">
                        <varName>N</varName>
                        <varType>int</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </terminates>
              </stateCondition>
            </atomicCondition>
            <WrappedCondition>
              <operator>and</operator>
              <assertionCondition>
                <atomicCondition conditionID="gt12-ac4">
                  <stateCondition>
                    <initiates>
                      <event>
                        <eventID forMatching="true" persistent="false">
                          <varName>VID0</varName>
                        </eventID>
                        <call>
                          <interfaceId>CP</interfaceId>
                          <OperationId>1</OperationId>
                          <operationName>resolve</operationName>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>status1</varName>
                            <varType>OpStatus</varType>
                            <value>REQ-B</value>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>sender1</varName>
                            <varType>Entity</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
                            <varName>receiver1</varName>
                            <varType>Entity</varType>
                          </inputVariable>
                          <inputVariable forMatching="true" persistent="false">
```

```
            <varName>source1</varName>
            <varType>Entity</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serviceId</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </inputVariable>
        </call>
        <tVar>
          <timeVar>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </timeVar>
        </tVar>
        <fromTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </fromTime>
        <toTime>
          <time>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
          </time>
        </toTime>
      </event>
      <state name="RepeatedResReq">
        <argument>
          <variable forMatching="true" persistent="false">
            <varName>seqrq</varName>
            <varType>int</varType>
          </variable>
        </argument>
        <argument>
          <operationCall>
            <name>add</name>
            <partner>self</partner>
            <variable forMatching="true" persistent="false">
              <varName>N</varName>
              <varType>int</varType>
            </variable>
            <constant>
              <name>Counter</name>
              <value>1</value>
            </constant>
          </operationCall>
        </argument>
      </state>
      <timeVar>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
```

```
                    </timeVar>
                  </initiates>
                </stateCondition>
              </atomicCondition>
            </assertionCondition>
          </WrappedCondition>
        </postcondition>
      </Guaranteed>
    </Assertion>
  </Anomalies>
</AssessmentScheme>
<ValidityTests/>
<MonitoringConfigurations>
  <MonitoringConfiguration Id="MC1">
    <Component>
      <Reasoner>
        <EndPoint>everestReasoner</EndPoint>
        <AssertionId>AS001</AssertionId>
      </Reasoner>
    </Component>
  </MonitoringConfiguration>
</MonitoringConfigurations>
<EvidenceAggregation>
  <AggregatedResultsInfo intervalUnit="days" intervalsTime="30"
                         Timestamp="2014-07-01" Startdate="2014-06-01"/>
  <FunctionalAggregatorId>Max</FunctionalAggregatorId>
  <IntermediateResults>false</IntermediateResults>
</EvidenceAggregation>
<LifeCycleModel>
  <InitialState stateId="is" name="Activated"/>
  <states>
    <state>
      <atomicState stateId="at1" name="Rejected"/>
    </state>
    <state>
      <compositeState stateId="cs1" name="ContinuousMonitoring">
        <substate>
          <compositeState stateId="cs2" name="Issuing">
            <substate>
              <atomicState stateId="as1" name="Pres-Issued"/>
            </substate>
            <substate>
              <atomicState stateId="as2" name="Issued/>
            </substate>
          </compositeState>
        </substate>
        <substate>
          <atomicState stateId="at2" name="Anomaly-Audit"/>
        </substate>
        <substate>
          <atomicState stateId="at3" name="Conflict-Audit/>
        </substate>
      </compositeState>
    </state>
  </states>
  <transitions>
```

```
        <transition From="is" To="at1">
          <WhenCondition>
            <event>assertionViolated</event>
          </WhenCondition>
        </transition>
        <transition From="is" To="as1">
          <WhenCondition>
            <event>sufficiencyConditionSatisfied</event>
          </WhenCondition>
        </transition>
        <transition From="as1" To="as2">
          <WhenCondition>
            <event>assertionSatisfied</event>
          </WhenCondition>
        </transition>
        <transition From="cs1" To="at1">
          <WhenCondition>
            <event>assertionViolated</event>
          </WhenCondition>
        </transition>
        <transition From="cs1" To="fs">
          <WhenCondition>
            <event>expirationReached</event>
          </WhenCondition>
        </transition>
        <transition From="cs2" To="at2">
          <WhenCondition>
            <event>anomalyDetected</event>
          </WhenCondition>
        </transition>
        <transition From="cs2" To="at3" >
          <WhenCondition>
            <event>conflictDetected</event>
          </WhenCondition>
        </transition>
        <transition From="at2" To="hs">
          <WhenCondition>
            <event>anomalyResolved</event>
          </WhenCondition>
        </transition>
        <transition From="at3" To="hs">
          <WhenCondition>
            <event>conflictResolved</event>
          </WhenCondition>
        </transition>
      </transitions>
      <FinalState stateId="fs" name="Revoked" />
      <historyState stateId="hs" name="history" refersToStateId="cs2" />
    </LifeCycleModel>
  </p:CertificationModel>
```

APPENDIX B: AUTHENTICATION CERTIFICATION MODEL

Below the security property element of the CM for the e-Health scenario is given, as explained in Section 5.3.

```
<SecurityProperty SecurityPropertyId="0001"
   SecurityPropertyDefinition="AIS:authentication:network-authenticated-server-access"
    Vocabulary="CSA" ShortName="http-to-http-redirection">
  <sProperty class="http://www.cumulus-project.eu">
    <propertyPerformance>
      <propertyPerformanceRow>
        <propertyPerformanceCell name="verified">"true" </propertyPerformanceCell>
      </propertyPerformanceRow>
    </propertyPerformance>
  </sProperty>
  <Assertion ID="AS001">
    <InterfaceDeclr>
      <ID>0001</ID>
      <ProviderRef>proRef1</ProviderRef>
      <Endpoint>
        <ID>eop1</ID>
        <Location>http://</Location>
        <Protocol>SOAP</Protocol>
      </Endpoint>
      <Interface>
        <InterfaceSpec>
          <Name>serverHTTP</Name>
          <Operation>
          <interfaceId>serverHTTP</interfaceId>
          <OperationId>id0001</OperationId>
          <operationName>httpCall</operationName>
          <inputVariable forMatching="true" persistent="false">
            <varName>format1</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serverAddress1</varName>
            <varType>string</varType>
          </inputVariable>
          <outputVariable>
            <varName>httpStatus2</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable>
            <varName>location2</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable>
            <varName>serverAddress2</varName>
            <varType>string</varType>
```

```
          </outputVariable>
        </Operation>
      </InterfaceSpec>
    </Interface>
  </InterfaceDeclr>
  <InterfaceDeclr>
    <ID>002</ID>
    <ProviderRef>proRef2</ProviderRef>
    <Endpoint>
      <ID>eop2</ID>
      <Location>http://</Location>
      <Protocol>SOAP</Protocol>
    </Endpoint>
    <Interface>
      <InterfaceSpec>
        <Name>serverHTTPS</Name>
        <Operation>
          <interfaceId>serverHTTPS</interfaceId>
          <OperationId>id0002</OperationId>
          <operationName>httpsCall</operationName>
          <inputVariable forMatching="true" persistent="false">
            <varName>format2</varName>
            <varType>string</varType>
          </inputVariable>
          <inputVariable forMatching="true" persistent="false">
            <varName>serverAddress3</varName>
            <varType>string</varType>
          </inputVariable>
          <outputVariable>
            <varName>format3</varName>
            <varType>string</varType>
          </outputVariable>
          <outputVariable forMatching="true" persistent="false">
            <varName>serverAddress4</varName>
            <varType>string</varType>
          </outputVariable>
          </Operation>
        </InterfaceSpec>
      </Interface>
    </InterfaceDeclr>
    <Guaranteed ID="gt1" type="Future_Formula">
      <quantification>
        <quantifier>forall</quantifier>
        <timeVariable>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVariable>
      </quantification>
      <quantification>
        <quantifier>existential</quantifier>
        <timeVariable>
          <varName>t2</varName>
          <varType>TimeVariable</varType>
        </timeVariable>
      </quantification>
      <precondition>
```

```
<atomicCondition conditionID="ac0">
  <eventCondition unconstrained="true">
    <event>
      <eventID forMatching="true" persistent="false">
        <varName>VID0</varName>
      </eventID>
      <call>
        <interfaceId>serverHTTP</interfaceId>
        <OperationId>1</OperationId>
        <operationName>httpCall</operationName>
        <inputVariable forMatching="true" persistent="false">
          <varName>status1</varName>
          <varType>OpStatus</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>sender1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>receiver1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>source1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>serviceId</varName>
          <varType>string</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>format1</varName>
          <varType>string</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>serverAddress1</varName>
          <varType>string</varType>
        </inputVariable>
      </call>
      <tVar>
        <timeVar>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVar>
      </tVar>
      <fromTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
      </fromTime>
      <toTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
```

```
            </toTime>
          </event>
        </eventCondition>
      </atomicCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition>
            <relationalCondition>
              <equal>
                <operand1>
                  <variable>
                    <varName>serverAddress1</varName>
                    <varType>string</varType>
                  </variable>
                </operand1>
                <operand2>
                  <constant>
                    <name>port80</name>
                    <value>8080</value>
                  </constant>
                </operand2>
              </equal>
              <timeVar>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </relationalCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="ac1">
        <eventCondition unconstrained="true">
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID1</varName>
            </eventID>
            <reply>
              <interfaceId>serverHTTP</interfaceId>
              <OperationId>2</OperationId>
              <operationName>httpCall</operationName>
              <outputVariable forMatching="true" persistent="false">
                <varName>status2</varName>
                <varType>OpStatus</varType>
                <value>RES-B</value>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>receiver1</varName>
                <varType>Entity</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>sender1</varName>
                <varType>Entity</varType>
              </outputVariable>
```

```
            <outputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </outputVariable>
            <outputVariable>
              <varName>httpStatus2</varName>
              <varType>string</varType>
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>location2</varName>
              <varType>string</varType>
            </outputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>serverAddress2</varName>
              <varType>string</varType>
            </outputVariable>
          </reply>
          <tVar>
            <timeVar>
              <varName>t2</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </fromTime>
          <toTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
            <Expression>
              <plus>1000</plus>
            </Expression>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
    <WrappedCondition>
      <operator>and</operator>
      <assertionCondition>
        <atomicCondition>
          <relationalCondition>
            <equal>
              <operand1>
                <variable>
                  <varName>location2</varName>
                  <varType>string</varType>
                </variable>
```

```
                </operand1>
                <operand2>
                  <constant>
                    <name>locationHttps</name>
                    <value>https://servername/</value>
                  </constant>
                </operand2>
              </equal>
              <timeVar>
                <varName>t2</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </relationalCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition>
            <relationalCondition>
              <equal>
                <operand1>
                  <variable>
                   <varName>httpStatus2</varName>
                   <varType>string</varType>
                  </variable>
                </operand1>
                <operand2>
                  <constant>
                    <name>RedirectStatus</name>
                    <value>301</value>
                  </constant>
                </operand2>
              </equal>
              <timeVar>
                <varName>t2</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </relationalCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </postcondition>
  </Guaranteed>
  <Guaranteed ID="gt2" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <timeVariable>
        <varName>t3</varName>
        <varType>TimeVariable</varType>
      </timeVariable>
    </quantification>
    <precondition>
      <atomicCondition conditionID="ac2">
        <eventCondition unconstrained="true">
```

```
<event>
  <eventID forMatching="true" persistent="false">
    <varName>VID2</varName>
  </eventID>
  <reply>
    <interfaceId>serverHTTPS</interfaceId>
    <OperationId>3</OperationId>
    <operationName>httpsCall</operationName>
    <outputVariable forMatching="true" persistent="false">
      <varName>status1</varName>
      <varType>OpStatus</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>sender1</varName>
      <varType>Entity</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>receiver1</varName>
      <varType>Entity</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>source1</varName>
      <varType>Entity</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>serviceId</varName>
      <varType>string</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>format3</varName>
      <varType>string</varType>
    </outputVariable>
    <outputVariable forMatching="true" persistent="false">
      <varName>serverAddress4</varName>
      <varType>string</varType>
    </outputVariable>
  </reply>
  <tVar>
    <timeVar>
      <varName>t3</varName>
      <varType>TimeVariable</varType>
    </timeVar>
  </tVar>
  <fromTime>
    <time>
      <varName>t3</varName>
      <varType>TimeVariable</varType>
    </time>
  </fromTime>
  <toTime>
    <time>
      <varName>t3</varName>
      <varType>TimeVariable</varType>
    </time>
  </toTime>
</event>
```

```
          </eventCondition>
        </atomicCondition>
        <WrappedCondition>
          <operator>and</operator>
          <assertionCondition>
            <atomicCondition>
              <relationalCondition>
                <equal>
                  <operand1>
                    <variable>
                      <varName>serverAddress4</varName>
                      <varType>string</varType>
                    </variable>
                  </operand1>
                  <operand2>
                    <constant>
                      <name>port443</name>
                      <value>443</value>
                    </constant>
                  </operand2>
                </equal>
                <timeVar>
                  <varName>t3</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </relationalCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </precondition>
      <postcondition>
        <atomicCondition conditionID="ac2">
          <relationalCondition>
            <equal>
              <operand1>
                <variable>
                  <varName>format3</varName>
                  <varType>string</varType>
                </variable>
              </operand1>
              <operand2>
                <constant>
                  <name>formatSSL</name>
                  <value>SSL-TLS</value>
                </constant>
              </operand2>
            </equal>
            <timeVar>
              <varName>t3</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </relationalCondition>
        </atomicCondition>
      </postcondition>
    </Guaranteed>
  </Assertion>
```

```
</SecurityProperty>
```

APPENDIX C: CONFIDENTIALITY CERTIFICATION MODEL

Below the security property element of the CM for the Smart Cities scenario is given, as explained in Section 5.4.

```
<SecurityProperty SecurityPropertyId="0001"
   SecurityPropertyDefinition="AIS:confidentiality:external-data-exchange-
   confidentiality:VPN" Vocabulary="CSA"
   ShortName="AIS:confidentiality:external-data-exchange-confidentiality">
  <sProperty class="http://www.cumulus-project.eu">
    <propertyPerformance>
      <propertyPerformanceRow>
        <propertyPerformanceCell name="verified">true</propertyPerformanceCell>
      </propertyPerformanceRow>
    </propertyPerformance>
  </sProperty>
  <Assertion ID="AS001">
    <InterfaceDeclr>
      <ID>0001</ID>
      <ProviderRef>proRef1</ProviderRef>
      <Endpoint>
        <ID>eop1</ID>
        <Location>http://</Location>
        <Protocol>SOAP</Protocol>
      </Endpoint>
      <Interface>
        <InterfaceSpec>
          <Name>ucapi0010</Name>
          <Operation>
            <interfaceId>ucapi0010</interfaceId>
            <OperationId>id0001</OperationId>
            <operationName>LCUCall</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>packetDest</varName>
              <varType>string</varType>
            </inputVariable>
            <outputVariable forMatching="true" persistent="false">
              <varName>packetSource</varName>
              <varType>string</varType>
            </outputVariable>
          </Operation>
        </InterfaceSpec>
      </Interface>
    </InterfaceDeclr>
```

```
<VariableDeclr>
  <varName>VPNaddressCall</varName>
  <varType>string</varType>
  <value>10.9.8.1:10022</value>
</VariableDeclr>
<VariableDeclr>
  <varName>VPNaddressReply</varName>
  <varType>string</varType>
  <value>10.9.8.10:10022</value>
</VariableDeclr>
<Guaranteed ID="gt1" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="ac0">
      <eventCondition unconstrained="true">
       <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>LCU</interfaceId>
            <OperationId>1</OperationId>
            <operationName>LCUCall</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>packetDest</varName>
              <varType>string</varType>
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
```

```
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </tVar>
            <fromTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </fromTime>
            <toTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </toTime>
          </event>
        </eventCondition>
      </atomicCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="as1">
            <stateCondition>
              <holdsAt>
                <state name="vVPNaddressCall">
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>VPNaddress</varName>
                      <varType>string</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </holdsAt>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="as2">
        <relationalCondition>
          <equal>
            <operand1>
              <variable forMatching="true" persistent="false">
                <varName>packetDest</varName>
                <varType>string</varType>
              </variable>
            </operand1>
            <operand2>
              <variable forMatching="true" persistent="false">
                <varName>VPNaddressCall</varName>
```

```
                <varType>string</varType>
              </variable>
            </operand2>
          </equal>
          <timeVar>
           <varName>t1</varName>
           <varType>TimeVariable</varType>
          </timeVar>
        </relationalCondition>
      </atomicCondition>
    </postcondition>
  </Guaranteed>
  <Guaranteed ID="gt2" type="Future_Formula">
    <quantification>
      <quantifier>forall</quantifier>
      <timeVariable>
        <varName>t1</varName>
        <varType>TimeVariable</varType>
      </timeVariable>
    </quantification>
    <precondition>
      <atomicCondition conditionID="ac0">
        <eventCondition unconstrained="true">
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID0</varName>
            </eventID>
            <reply>
              <interfaceId>LCU</interfaceId>
              <OperationId>1</OperationId>
              <operationName>LCUCall</operationName>
              <outputVariable forMatching="true" persistent="false">
                <varName>status1</varName>
                <varType>OpStatus</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>sender1</varName>
                <varType>Entity</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>receiver1</varName>
                <varType>Entity</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>source1</varName>
                <varType>Entity</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>serviceId</varName>
                <varType>string</varType>
              </outputVariable>
              <outputVariable forMatching="true" persistent="false">
                <varName>packetSource</varName>
                <varType>string</varType>
              </outputVariable>
            </reply>
```

```
            <tVar>
              <timeVar>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </tVar>
            <fromTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </fromTime>
            <toTime>
              <time>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </time>
            </toTime>
          </event>
        </eventCondition>
      </atomicCondition>
      <WrappedCondition>
        <operator>and</operator>
        <assertionCondition>
          <atomicCondition conditionID="as1">
            <stateCondition>
              <holdsAt>
                <state name="vVPNaddressReply">
                  <argument>
                    <variable forMatching="true" persistent="false">
                      <varName>VPNaddressReply</varName>
                      <varType>string</varType>
                    </variable>
                  </argument>
                </state>
                <timeVar>
                  <varName>t1</varName>
                  <varType>TimeVariable</varType>
                </timeVar>
              </holdsAt>
            </stateCondition>
          </atomicCondition>
        </assertionCondition>
      </WrappedCondition>
    </precondition>
    <postcondition>
      <atomicCondition conditionID="as2">
        <relationalCondition>
          <equal>
            <operand1>
              <variable forMatching="true" persistent="false">
                <varName>packetSource</varName>
                <varType>string</varType>
              </variable>
            </operand1>
            <operand2>
```

```
            <variable forMatching="true" persistent="false">
              <varName>VPNaddressReply</varName>
              <varType>string</varType>
            </variable>
          </operand2>
        </equal>
        <timeVar>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVar>
      </relationalCondition>
    </atomicCondition>
  </postcondition>
</Guaranteed>
<Guaranteed ID="gt3" type="Future_Formula">
  <quantification>
    <quantifier>forall</quantifier>
    <timeVariable>
      <varName>t1</varName>
      <varType>TimeVariable</varType>
    </timeVariable>
  </quantification>
  <precondition>
    <atomicCondition conditionID="ac0">
      <eventCondition unconstrained="true">
        <event>
          <eventID forMatching="true" persistent="false">
            <varName>VID0</varName>
          </eventID>
          <call>
            <interfaceId>VPN_server</interfaceId>
            <OperationId>1</OperationId>
            <operationName>setupVPN</operationName>
            <inputVariable forMatching="true" persistent="false">
              <varName>status1</varName>
              <varType>OpStatus</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>sender1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>receiver1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>source1</varName>
              <varType>Entity</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>serviceId</varName>
              <varType>string</varType>
            </inputVariable>
            <inputVariable forMatching="true" persistent="false">
              <varName>setupVPNaddress</varName>
              <varType>string</varType>
```

```
            </inputVariable>
          </call>
          <tVar>
            <timeVar>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </tVar>
          <fromTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </fromTime>
          <toTime>
            <time>
              <varName>t1</varName>
              <varType>TimeVariable</varType>
            </time>
          </toTime>
        </event>
      </eventCondition>
    </atomicCondition>
    <WrappedCondition>
     <operator>and</operator>
      <assertionCondition>
        <atomicCondition conditionID="ac1">
          <stateCondition>
            <holdsAt>
              <state name="vVPNaddress">
                <argument>
                  <variable forMatching="true" persistent="false">
                    <varName>VPNaddress</varName>
                    <varType>string</varType>
                  </variable>
                </argument>
              </state>
              <timeVar>
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </holdsAt>
          </stateCondition>
        </atomicCondition>
      </assertionCondition>
    </WrappedCondition>
  </precondition>
  <postcondition>
    <atomicCondition conditionID="as1">
      <stateCondition>
        <terminates>
          <event>
            <eventID forMatching="true" persistent="false">
              <varName>VID0</varName>
            </eventID>
            <call>
```

```
      <interfaceId>VPN_server</interfaceId>
        <OperationId>1</OperationId>
        <operationName>setupVPN</operationName>
        <inputVariable forMatching="true" persistent="false">
          <varName>status1</varName>
          <varType>OpStatus</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>sender1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>receiver1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>source1</varName>
          <varType>Entity</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>serviceId</varName>
          <varType>string</varType>
        </inputVariable>
        <inputVariable forMatching="true" persistent="false">
          <varName>setupVPNaddress</varName>
          <varType>string</varType>
        </inputVariable>
      </call>
      <tVar>
        <timeVar>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </timeVar>
      </tVar>
      <fromTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
      </fromTime>
      <toTime>
        <time>
          <varName>t1</varName>
          <varType>TimeVariable</varType>
        </time>
      </toTime>
    </event>
    <state name="vVPNaddress">
      <argument>
        <variable forMatching="true" persistent="false">
          <varName>VPNaddress</varName>
          <varType>string</varType>
        </variable>
      </argument>
    </state>
    <timeVar>
```

```
                <varName>t1</varName>
                <varType>TimeVariable</varType>
              </timeVar>
            </terminates>
          </stateCondition>
        </atomicCondition>
        <WrappedCondition>
          <operator>and</operator>
          <assertionCondition>
            <atomicCondition conditionID="as2">
              <stateCondition>
                <initiates>
                  <event>
                    <eventID forMatching="true" persistent="false">
                    <varName>VID0</varName>
                  </eventID>
                  <call>
                   <interfaceId>VPN_server</interfaceId>
                   <OperationId>1</OperationId>
                   <operationName>setupVPN</operationName>
                   <inputVariable forMatching="true" persistent="false">
                      <varName>status1</varName>
                      <varType>OpStatus</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>sender1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>receiver1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>source1</varName>
                      <varType>Entity</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>serviceId</varName>
                      <varType>string</varType>
                    </inputVariable>
                    <inputVariable forMatching="true" persistent="false">
                      <varName>setupVPNaddress</varName>
                      <varType>string</varType>
                    </inputVariable>
                  </call>
                  <tVar>
                    <timeVar>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </timeVar>
                  </tVar>
                  <fromTime>
                    <time>
                      <varName>t1</varName>
                      <varType>TimeVariable</varType>
                    </time>
```

```
                    </fromTime>
                    <toTime>
                      <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                      </time>
                    </toTime>
                  </event>
                  <state name="vVPNaddress">
                    <argument>
                      <variable forMatching="true" persistent="false">
                      <varName>setupVPNaddress</varName>
                        <varType>string</varType>
                      </variable>
                    </argument>
                  </state>
                  <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                  </timeVar>
                </initiates>
              </stateCondition>
            </atomicCondition>
          </assertionCondition>
        </WrappedCondition>
      </postcondition>
    </Guaranteed>
  </Assertion>
</SecurityProperty>
```

APPENDIX D: QUESTIONNAIRE

Questionnaire for certifies to evaluate the monitoring based certification model.

Evaluation Questions for Monitoring based certification process

1. Do you think that the Monitoring Based Certification Models (MBCM) are capable of representing comprehensively continuous security certification processes for cloud services security? (select only the answer that fits best with your view):

   0: No, not at all [ ]
   1: Yes, but in less than 25% of cases that I can think of [ ]
   2: Yes, in about 25-49 % of cases that I can think of [ ]
   3: Yes, in about 50-74 % of cases that I can think of [ ]
   4: Yes, in about 75-90 % of cases that I can think of [ ]
   5: Yes, in excess of 90% of cases that I can think of [ ]

   If you selected 4 or less, can you give examples of security properties and/or cloud services, which cannot be effectively certified based on the MBCMs?

2. Do you think that the assertion rules specified as part of a the MBCM are capable of representing accurately and effectively the continuous collection of evidence required for the assessment of security properties and/or the effectiveness of control mechanisms realising these properties in the cloud? (select only the answer that fits best with your view):

   0: No, not at all [ ]
   1: Yes, but in less than 25% of cases that I can think of [ ]
   2: Yes, in about 25-49 % of cases that I can think of [ ]
   3: Yes, in about 50-74 % of cases that I can think of [ ]
   4: Yes, in about 75-90 % of cases that I can think of [ ]
   5: Yes, in excess of 90% of cases that I can think of [ ]

   If you selected 4 or less, can you give examples of security properties/control mechanisms that cannot be effectively monitored by the assertions in MBCMs?

3. Do you think that the life cycle models specified as part of a MBCM are capable of representing effectively the processes of collecting evidence, and generating and managing certificates based on it? (select only the answer that fits best with your view):

0: No, not at all	[ ]
1: Yes, but in less than 25% of cases that I can think of	[ ]
2: Yes, in about 25-49 % of cases that I can think of	[ ]
3: Yes, in about 50-74 % of cases that I can think of	[ ]
4: Yes, in about 75-90 % of cases that I can think of	[ ]
5: Yes, in excess of 90% of cases that I can think of	[ ]

If you selected 4 or less, can you give examples of cases which life cycle models would not be adequate for?

4. Do you think that the evidence sufficiency conditions that may be specified as part of a MBCM (number of events, period of monitoring, expected behaviour of target of certification) are capable of representing effectively the circumstances under which the evidence collected would be enough to make a decision about issuing a certificate or otherwise? (select only the answer that fits best with your view):

0: No, not at all	[ ]
1: Yes, but in less than 25% of cases that I can think of	[ ]
2: Yes, in about 25-49 % of cases that I can think of	[ ]
3: Yes, in about 50-74 % of cases that I can think of	[ ]
4: Yes, in about 75-90 % of cases that I can think of	[ ]
5: Yes, in excess of 90% of cases that I can think of	[ ]

If you selected 4 or less, can you give examples of cases which evidence sufficiency conditions would not be adequate for?

5. Which of the following parts of MBCMs, do you think that it would be difficult for someone with expertise in cloud security to specify even after training (select all that apply):

None	[ ]
Assertions expressing the collection of evidence for security properties/anomalies	[ ]
Evidence sufficiency conditions	[ ]
Life cycle models	[ ]

6. Are there any key elements/requirements that continuous security certification processes for cloud services should address but MBCM fail to cover?

No	[ ]
Yes	[ ]

If “YES”, please indicate the missing elements/requirements: