Project Acronym: CUMULUS
Project Title: Certification infrastrUcture for MUlti-Layer cloUd Services
Call identifier: FP7-ICT-2011-8
Grant agreement no.: 318580
Starting date: 1st October 2012
Ending date: 30th September 2015

D2.3 Certification models v.2

AUTHOR(S): Ernesto Damiani (UMIL), George Spanoudakis (CITY), Spyros Katopodis (CITY), Khaled Mahbub(CITY), Maria Krotsiani (CITY), Stelvio Cimato (UMIL), Marco Anisetti (UMIL), Claudio Ardagna (UMIL), Francesco Zavatarelli (UMIL), Maria Rosa Vieira Alvarez (ATOS), Renato Menicocci (FUB), Alessandro Riccardi (FUB), Vittorio Bagini(FUB),Javier Espinar (UMA), Antonio Muñoz (UMA), Antonio Maña (UMA), Hristo Koshutanski (UMA)

REVIEWERS(S): Alain Pannetrat (CSA), Daniel Schmoelzer (IFX), Matthias Junk (IFX)

Date: May 30, 2014

Summary

Executive Summary **5**
1. Introduction **6**
2. A modular Certification Meta-Model **10**
2.1. General consideration on CUMULUS Meta-Model *10*
2.2. Naming convention *12*
2.3. Property vocabulary module *12*
2.4. Certificate module *14*
2.5. Certification Model module *15*
2.6. Certification Process module *17*
2.6.1. Chain of Trust 18
3. CUMULUS Meta-Model compared with a Common Criteria Model **21**
3.1. CC Model description *21*
3.2. CC Model mapping onto CUMULUS Meta-Model *23*
4. Test-based Certification Model **27**
4.1. Test-based Certification Model XML Schema Description *27*
4.1.1. Certification Model Id 29
4.1.2. Life Cycle 30
4.1.3. Target of Certification (ToC) Element 34
4.1.4. SecurityProperty Element 35
4.1.5. Signature Element 38
4.1.6. Collectors Element 39
4.1.7. Aggregator Sub-Element 42
4.1.8. Context Element 43
4.2. Test-based Certificate XML Schema *44*
4.2.1. Assertion 46
5. Monitoring-based Certification Model **47**
5.1. Overview *47*
5.2. Non repudiation of cloud storage services: an example of a service to be certified *50*
5.3. Certification Model XML Schema Description *53*
5.3.1. Model_Id Element 54
5.3.2. CASignature Element 54
5.3.3. TargetOfCertification (TOC) Element 54
5.3.4. SecurityProperty Element 57
5.3.5. AssessmentScheme Element 67
5.3.6. StateTransition Model 73
5.3.7. LogicalExpressionType Element 81
5.3.8. AnomalyMonitoring Element 87
5.3.9. ValidityTests Element 103
5.3.10. MonitoringConfigurations Element 103
5.3.11. EvidenceAggregation Element 105
5.3.12. LifeCycleModel Element 106
6. TC-based Certification Model **112**
6.1. TC-based Certification Model XML Schema Description *112*
6.1.1. Model_Id Element 114
6.1.2. TCCertificateType Element 114
6.1.3. CASignature Element 114
6.1.4. Security Property 114
6.1.5. AssessmentScheme Element 114
6.1.6. ValidityTests Element 114
6.1.7. EvidenceAggregation Element 114
6.1.8. LifeCycleModel Element 115
6.2. TC Support for Certification *115*

6.2.1. TC Support Basics 115
6.2.2. TC Support Model 116
7. Advanced Certification Models 120
7.1. Multi-layer certification 120
7.1.1. Multi-layer Test-based certification 120
7.1.2. Multi-layer Monitoring-based certification 124
7.2. Incremental certification 124
7.2.1. Incremental Test-based certification 124
7.2.2. Incremental Monitoring-based certification 126
7.3. Hybrid certification 126
7.3.1. Comparison between hybrid and traditional certification models 151
8. Conclusion 152
9. References 153
10. Appendix 155
10.1. Test-based Certification Model generic schema (.xsd file) 155
10.2. Test-based Certification Model instance (.xml file) 161
10.3. Monitoring-based Certification Model generic schema (.xsd file) 169
10.4. Monitoring-based Certification Model instance (.xml file) 178
10.5. TC Support for Certification Schema (.xsd file) 191
10.6. Specification of the BNF grammar of SecureSLA 192*

List of Figures

Figure 1 – CUMULUS Meta-Model 11
Figure 2 – Instantiation and Naming Convention 12
Figure 3 – Cloud Layers, ToC, Cloud component instances and certificate binding (single layer) 16
Figure 4 – Example of Certificate Life Cycle 16
Figure 5 – Definition Process: Models and Instances 18
Figure 6 – CoT (Chain of Trust) 19
Figure 7 – Signature process 19
Figure 8 – Common Criteria model 22
Figure 9 – Test-based Certification Model 28
Figure 10 – Test-based CM: Cm Id Element Type 29
Figure 11 – Test-based CM: Life Cycle Element Type 30
Figure 12 - Test-based CM: Life Cycle State Transitions 31
Figure 13 – Test-based CM: Life Cycle Transition Type 31
Figure 14 – Test-based CM: TOC Element Type 34
Figure 15 – Test-based CM: Security Property Element Type 36
Figure 16 – Test-based CM: Security Property Sub-Element Type 36
Figure 17 – Test-based CM: Signature Element Type 38
Figure 18 – Test-based CM: Collectors Element Type 39
Figure 19 – Test-based CM: Abstract Collector Sub-Element 39
Figure 20 – Test-based CM: Collector Sub-Element 40
Figure 21 – Test-based CM: Event Bus Collector Sub-Element 41
Figure 22 – Test-based CM: Aggregator Sub-Element 43
Figure 23 – Test-based CM: Context Element Type 44
Figure 24 – Test-based Certificate 45
Figure 25 – Test-based Certificate: Certificate Info Type 45
Figure 26 – Test-based Certificate: Assertion 46
Figure 27 – Non Repudiation protocol for cloud storage services (based on (Feng, 2011)) 51

Figure 28 – Monitoring-based Certification Model schema elements........53
Figure 29 – Monitoring-based CM: Target Of Certification Type........55
Figure 30 – Monitoring-based CM: ProvidesInterface Type........56
Figure 31 – Monitoring-based CM: RequiresInterface Type........56
Figure 32 – Monitoring-based CM: Assertion Element Type........58
Figure 33 – Monitoring-based CM: Interface Declaration Type........61
Figure 34 – Monitoring-based CM: Variable Declaration Type........64
Figure 35 – Monitoring-based CM: Guaranteed State Type........65
Figure 36 – Monitoring-based CM: Assessment Scheme Type........68
Figure 37 – Monitoring-based CM: Evidence Sufficiency Condition Type........69
Figure 38 – Example of expected TOC Behaviour Model........70
Figure 39 – Monitoring-based CM: Expiration Condition Type........71
Figure 40 – Monitoring-based CM: Conflict Type........72
Figure 41 – Monitoring-based CM: State Transition Model Type........74
Figure 42 – Monitoring-based CM: Pseudo State Type........75
Figure 43 – Monitoring-based CM: History State Type........75
Figure 44 – Monitoring-based CM: Atomic State Type........76
Figure 45 – Monitoring-based CM: Operation Ref Type........77
Figure 46 – Monitoring-based CM: Composite States Type........78
Figure 47 – Monitoring-based CM: Transition Element........79
Figure 48 – Monitoring-based CM: Logical Expression Type and Condition Type........81
Figure 49 – Monitoring-based CM: Evidence Condition Type........83
Figure 50 – Monitoring-based CM: Arithmetic Expression Type........85
Figure 51 – Monitoring-based CM: Anomaly Type........88
Figure 52 – Monitoring-based CM: Validity Tests Type........103
Figure 53 – Monitoring-based CM: Individual Monitoring Configuration Type........104
Figure 54 – Monitoring-based CM: Evidence Aggregation Type........105
Figure 55 – Monitoring-based CM: Lifecycle Model Type........106
Figure 56 – Monitoring-based CM: UML diagram of Life Cycle Model........107
Figure 57 – TC Certification Model schema........113
Figure 58 – Binding of Testing/Monitoring Agent to a Platform state........115
Figure 59 – TC-Support Model schema........117
Figure 60 – TC-Support for Certification Example........119
Figure 61 – Storage service to be certified........121
Figure 62 – Dependent mode hybrid certification models........141

List of Tables

Table 1 - Status of sections........9
Table 2 – Certification Models........15
Table 3 – Certification Models Mapping between Common Criteria Model elements and CUMULUS Meta-Model classes........26
Table 4 – Mapping between Model classes and Meta-Model classes for the Test-based Certification Model.........28
Table 5 – Mapping of Monitoring Based Certification Models onto CUMULUS Meta-Model classes........49
Table 6 – TC Certification Model Schema........113

Executive Summary

This document is the second version of the conceptual framework underlying the specifications of *basic (Test-based, Monitor-based, Trusted Computing (TC)-based), multi-layer, hybrid* and *incremental* Certification Models for cloud-based processes, applications, and services, and the schemas used for specifying such models. The proposed approach relies on a unifying *meta-model* to provide representational guidelines for (i) the definition of the security properties to be certified (ii) the types of evidence underlying them (iii) the phases of the certificate life cycle, as well as of all mechanisms for generating supporting evidence. The first version of test, monitoring and trusted computing based certification models was introduced in deliverable D2.2. To provide a comprehensive and definitive version of the models, in this deliverable we document not only the changes to the first version but also the unaltered parts of it.

In summary, the second version of certification models has introduced the following changes:

- In the case of test based certification models, v2 v2 has completely redefined the model of the Target of Certification, including a concept of Target of Tests which represent the hooks to our testing probes, and the model of how evidence is collected and aggregated, redesigning the Collectors element in the Certification Model.
- In the case of monitoring based certification models, v2 has introduced models of expected behaviour for Targets of Certification, a new scheme for defining sufficiency conditions regarding monitoring evidence (conflict and sufficiency wrt expected behaviour), a new scheme for specifying anomalies that should be monitored (along with security properties) in generating monitoring based certificates, and a more elaborated scheme for specifying life cycle models for the production and management of test based certificates.
- In the case of trusted computing certification models, the main changes relate to to the definition of a TC support model for certification leveraging the trust necessary for validation of the monitoring and test based certifications. The TC support model is seen as crucial for the relaiability and acceptance of evidences gatherend by monitoring and testing agents.

Examples of Test-based, Monitoring-based and TC-based Certification Models and related Certificates are made. Then a preliminary version of *multi-layer, hybrid* and *incremental* Certification Models is proposed.

1. Introduction

The freedom of choice given by dynamic software provisioning on the cloud needs to be reconciled with the apparently conflicting requirement of making sure that software systems have the appropriate assurance levels for the intended purpose. In the case of security, influential guidelines like the US-NIST Special Publication (SP) 800-37s mandate users to check *at the time of use* that software systems hold the desired set of security properties. This requirement is very difficult to satisfy in a cloud-based service marketplace, and in general with any dynamic model of software provisioning.

According to the vision of the CUMULUS project, checking security properties is greatly facilitated when some accredited authority continuously produces, maintains and manages signed certificates regarding the properties held by cloud entities and applications, as well as the evidence supporting such properties.

In our approach, the certification process is a function that takes as input the assertions about a security property and a target of certification, and produces as output a machine-readable certificate containing the evidence that proves the requested properties, by means of verification and validation techniques.

There are different ways of persuading a customer that a security property holds for a given service. It is possible to produce test-based evidence, that is, evidence that a test carried out on the software has given a certain result (static or offline tests); alternatively, one could test dynamically when a specific service is invoked (dynamic tests), monitor the service continually, rely on the use and/or synthesis of certificates of components which contribute to the realisation of the service, or deploy hybrid schemes combining more than one of these different types of evidence. Note that changes in the service may affect the service model and in turn the evidence generated to support a given property.

A major effect of the certification process is that it modifies the trust of customer c in the security assertions made on a service instance. As discussed in (Anisetti A. &., 2011)the lack of a certification process for services results in a scenario where the level of trust Tc(Aws), with Aws a set of assertions made by the service provider on its web service ws, mainly depends on the service provider's reputation. With the advent of service certification, the trust model includes the trust of customer c in assertions made by a certification authority on a service, denoted as Tc(C), where C is the certificate awarded to the service ws by the certification authority. Trust Tc(C) considers the assertions, the properties, the model, and the evidence in the certificate. Certification is effective if Tc(C) ≥ Tc(Aws), meaning that the credibility of the certification authority is greater than the one of the service provider. The certification process permits to extend the service-based infrastructure with runtime selection of services based on security certificates and customer preferences. This certification, in fact, produces a set of metadata in the form of machine-readable certificates, which can be used to compare and rank services, and identify the best one addressing customer preferences. A client (i.e., a software agent acting on behalf of a human user or a service provider aiming to implement a business process) can then select and compose services at run time on the basis of the security properties and evidence in the certificate.

A major step toward implementing the CUMULUS vision is the proposition of a shared representation of the domain, i.e. defining a model including all conceptual entities involved in the certification of cloud entities. Such model lies at the core of the CUMULUS project, since how a certificate is produced, what is its content, and how it is managed are all questions of paramount importance for the project. Furthermore, both the certification infrastructure and the service engineering tools, that are goals of WP4 and WP5, respectively, need the basic definitions of a certificate model to define the requirements, the architecture and proceed towards the complete development of the whole framework.

As stated in the CUMULUS Description of Work (DoW), the conceptual framework developed in Work Package 2 will deal with the specifications of *basic*, *hybrid, multi-layer* and *incremental* certification models for cloud-based processes, applications, and services, defining the security properties of interest for certification, and the types of evidence used for certificate issuing as well as the relevant mechanisms for generating the evidence supporting a security property.

Tasks 2.2, 2.3 and 2.4 of WP2 have the goal of defining three basic certification models: one for test-based certification, i.e. where evidence is collected after the execution of test procedures; the second one for monitoring-based certification, where the evidence is collected by monitoring the service during its execution; and the last one for TC-based certification, where evidence is based on the lower layers of the cloud stack and on the function set of a Trusted Platform Module. In WP2, the certification model should also provide ways of combining different evidences in an integrated framework, paving the way for the definition of hybrid and incremental certificates. Incremental certification is particularly important: 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 that is an instance of this type can 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. This document discusses a generic life cycle model including the possible updates and other key changes in the life cycle of incremental certificates.

The first efforts have been directed towards the definition of a unifying *meta-model*, under which different certificate models can be treated. The meta-model gives an additional abstraction layer, allowing both the definition of the common characteristics of the different models and the possibility to define the instances, which is the detailed model for each different category of certificate. The vocabulary of terms, the entities and the detailed description of the meta-model are provided in Section 3.

Section 4 provides a first comparison (which will be further refined and analysed in future CUMULUS deliverables) between the abovementioned meta-model and a model of Common Criteria certification. This comparison mainly aims at assessing the alignment of the CUMULUS meta-model with a real world certification approach.

Sections 5, 6, and 7 contain the detailed description of the Test-based, Monitoring-based, and TC-based certification models, respectively, and examples of Test-based, Monitoring-based and TC-based Certification Models and related Certificates are made.

Section 8 has a preliminary version of *multi-layer, hybrid* and *incremental* Certification Models. Final versions will be provided in Deliverable D2.4, “Final CUMULUS certification models”, to be released at M32.

As prescribed in CUMULUS Document of Work (CUMULUS Consortium, DoW, 2012), this deliverable “Certification models v.2” is released at M20, and is the update of “Certification models v.1” released in M12. A further refinement of the Certification Models will be included in D2.4 “Final CUMULUS certification models”. To help the reader better understand the evolution of the deliverable and the underlying certification models which are documented by it, each section of the deliverable contains a brief description of the changes between v1 of the certification models (i.e., the version described in deliverable D2.2) and v2 of the certification (i.e., the version described in this deliverable). A summary of the main changes is also presented inTable 1 below.

This deliverable is also closely related to another deliverables of the CUMULUS project: D2.1 “Security-aware SLA specification language and cloud security dependency model” (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013), which defines vocabulary to be used in project and the syntax of security property definitions.

Section	Status	Modification
1 - Introduction		
2 - A modular Certification Meta-Model	Completed, some minor modifications may occur	The Metamodel was updated. The Certification Model module (section 2.6) has been fully developed with new concepts. Sub-section about chain of trust (section 2.6.1) is also new.
3 - CUMULUS Meta-Model compared with a Common Criteria Model	Completed, some minor modifications may occur	New section (not present in D2.2)
4 - Test-based Certification Model	Updated	The main modifications of the test based certification models are in the Target of Certification and in the Abstract Collectors elements. In the former element all the testing hooks, i.e. the accessible APIs for CUMULUS framework, are specified in a Target of Tests sub-element. The Abstract Collectors element instead defines the evidence collecting process, i.e. how evidence is collected, aggregated and archived.
5 - Monitoring-based Certification Model		The main modifications of the monitoring based certification models relate to the introduction of models of expected behaviour for Targets of Certification, the amendment of the scheme for defining sufficiency conditions regarding monitoring evidence, the introduction of a scheme for specifying anomalies that should be monitored (along with security properties) in generating monitoring based certificates, and the introduction of a more elaborated scheme for specifying life cycle models.
6 - TC-based Certification Model		A new TC support for certification model is presented that provides the trust necessary to validate test and monitoring based certification. The new model is leveraged upon the use of TPM v1.2 specification, especially the cryptographic key binding fucntionality allowing a testing/monitoring agent to sign

		evidence (data) only if the platform and the agent are in valid state. The model also considers restrictive and permissive TC supoprt depending on the flexibility one needs.
7 - Advanced Certification Models	Some initial ideas regarding the form of these models are presented in this deliverables. The models will be properly defined in deliverable D2.4	New section (not present in D2.2), covering hybrid and incremental certification
8 - Conclusion		
Appendix		

Table 1 - Status of sections

2. A modular Certification Meta-Model

In this section we describe the modular Meta-Model underlying the CUMULUS cloud certification scheme. The main reason of the Meta-Model definition is pragmatism: first we want to provide preliminary basic shared conceptualization to guide modellers of individual certification schemes and, secondly, given the complexity of a security environment which deals with heterogeneous stakeholders, we feel we do need to share some common understanding about key concepts, such as certification, certificates, security properties and assurance.

2.1. General consideration on CUMULUS Meta-Model

This section amends section 3.1 in D2.2. There are some changes in the proposed Meta Model: the Target of Certification is now linked to the Context and the Certificate has now a LifeCycle.

The aim of CUMULUS Meta-Model the aim is threefold:

(i) Modularity, enabling a clean subdivision of the CUMULUS framework into different modules

(ii) "Assertion centeredness", ensuring that all transient and context features are attached to assertions including security properties rather than to the properties themselves. This will make CUMULUS property definitions reusable and sharable.

(iii) Simplicity, that is keeping the Meta-Model simple, concise and easy to understand.

The Meta-Model shown in Figure 1 is designed to meet these requirements.

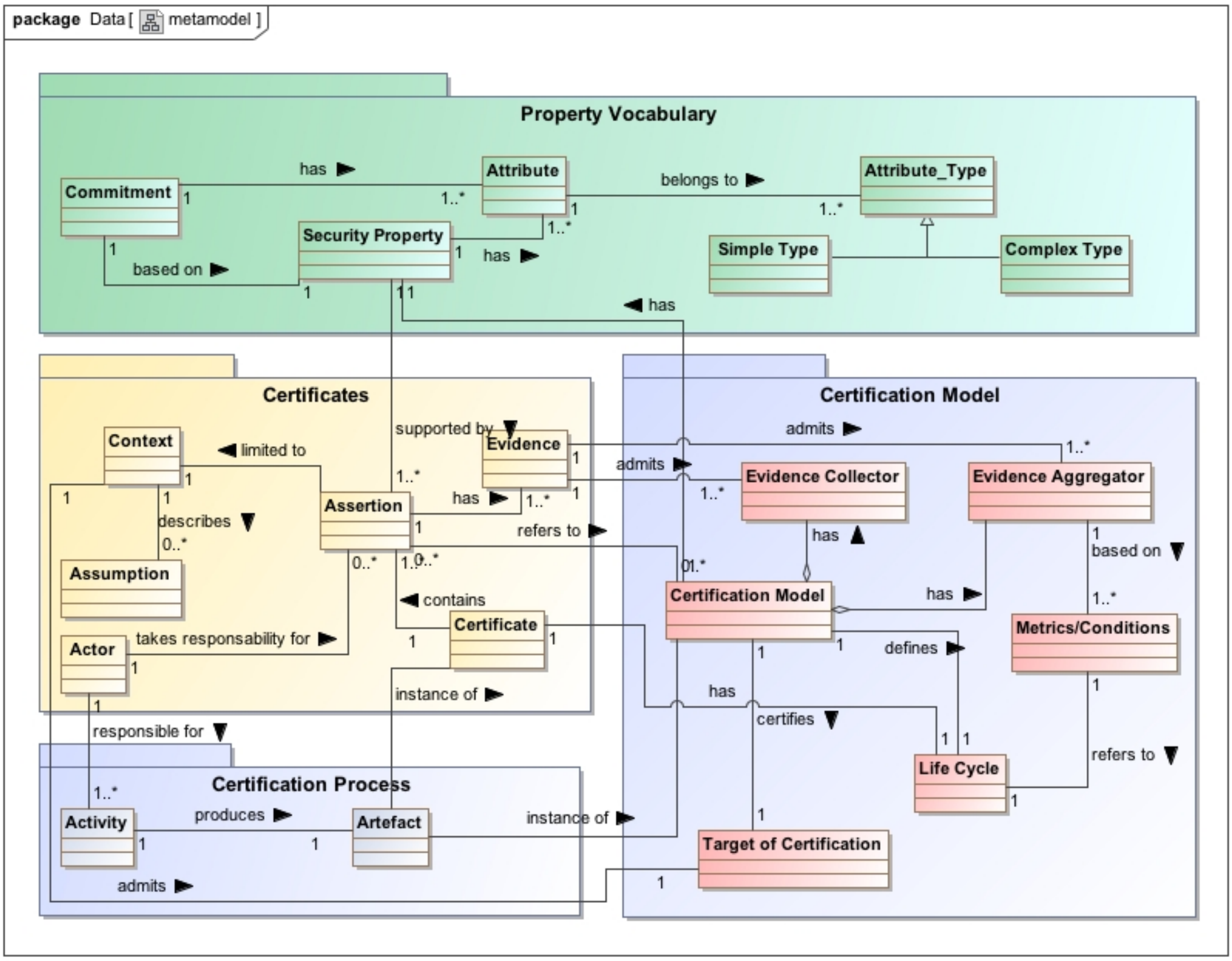


FIGURE 1 – CUMULUS META-MODEL

The CUMULUS Meta-Model has a modular structure representing: (i) the Property Vocabulary module with *Commitment*, *Security Properties* plus *Attributes* and *Types*, (ii) the Certificate module with *Certificate*, *Assertion*, *Evidence, Context, Assumptions* and *Actor* entities, (iii) the Certification Model module, including *Certification Model*, *Life Cycle*, *TOC*, *Evidence Collection, Evidence Aggregation* and *Metrics/Conditions* entities. The *Commitment* entity was not included in any module in a previous version of this deliverable, mainly because it has just the purpose to be a bridge toward future SLA definitions (Damiani, Ardagna, & Bezzi, 2012), but now it is in the Property Vocabulary module to give more consistency to the overall architecture. Another module called Certification Process module has been introduced to take into account the process that an Actor, as a Certification Authority, could undertake to produce Certification Model instances and Certificates. All the modules have been designed as independent components. Next, we briefly review the individual modules and their use.

For the sake of simplicity, our Meta-classes are represented via a UML class diagram. Our meta-classes represent guidelines for modellers, that is, they will be instantiated into model classes by modellers when they set up a certification model for a specific environment (Test, Monitoring, TC). Then, model classes will be instantiated in certification artefacts, i.e. Certification Models and Certificates. As general criteria, we leave the Meta-Model largely informal, and define the single models as semi-formal UML class diagrams. In order to provide guidelines for certificate production and processing, besides having a *Life Cycle* entity in the Meta-Model, we model the certificate life cycle as a UML state activity diagram. This choice goes in the direction of readability and ready applicability on the part of modellers.

Furthermore, we assume that:

- relations between Meta-Model entities are all associations and all entities can be instantiated independently, and
- relations between Meta-Model and Model entities are all an “is-a relationship” and will be represented as generic dependence in UML with a comment.

Besides providing guidelines to modellers, the Meta-Model defines some important (though loose) constraints on models: for instance, all *Assertion* instances contained in the *Certificate* should be of the types listed in the *Certificate Model*.

Security Property Meta-Model entities express abstract security properties; some abstractions can be defined over the Model classes, thus introducing a hierarchical multi-layered model for security properties.

2.2. Naming convention

This section corresponds to section 3.2 in D2.2. There are no amendments .

Since our Meta-Model is intended to support the design of certification models to be later instantiated into artefacts, it is useful to define a naming convention. In particular, we use a *sequence-based naming scheme* with the following notation: the name of the Meta-Model class becomes the final suffix and then starting from this suffix all the classes and sub classes of the Model tree are attached. Finally we have the instance.

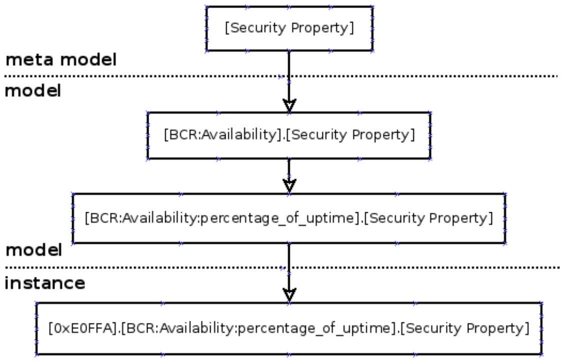

[Security Property]
meta model
model
[BCR:Availability].[Security Property]
[BCR:Availability:percentage_of_uptime].[Security Property]
model
instance
[0xE0FFA].[BCR:Availability:percentage_of_uptime].[Security Property]

FIGURE 2 – INSTANTIATION AND NAMING CONVENTION

For example a Model class is denoted by [Model Class Name].[Meta-Model Class Name], a Model sub class is indicate by [Sub Class Name.Model Class Name].[Meta-Model Class Name], finally an instance is [Instance ID].[Model Class Name].[Meta-Model Class Name]. For the overall sequence-based naming scheme, see Figure 2. For a Security Property entity the Model name takes the definition of Deliverable D2-1 (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013) where its full name is defined as a Cloud Control Matrix (CCM) control domain name (“AIS”) plus a sub-category (“integrity”) plus the name of the property (“data-alteration-detection”).

2.3. Property vocabulary module

This section corresponds to section 3.3 in D2.2. There are no amendments to this part.

The upper area of Figure 1 identifies the property vocabulary module and defines a very generic framework where *Security Property* and *Attribute* are linked in a 1:N relationship (i.e. a set of *Attributes* is associated to a *Security Property*) and allows a simple classification of the *Attributes* in terms of *Types*. We could say that a *Property* is an aggregation of *Attributes* and an *Attribute* can take a value in at least two domains, Simple and Complex. The Simple Type can be used to map XML data type, the Complex Type is meant for a modeller who wants to define a new type; there are no specific requirements for a new type except to be defined with a Complex Type as root.

The *Security Property* Meta-Model entities express abstract security properties: basic examples of abstract security properties are Confidentiality, Integrity, Authenticity, and Availability. It should be noted that this is not an exhaustive list of property types and in any case there is no consensus about them (Irvine & Levin, 1991) (Chung & Leite, Conceptual modeling, 2009) (Chung, Nixon, Yu, & Mylopoulos, 2000). In particular, the Assert4SOA project indicates the following abstract properties: Confidentiality, Integrity, Authentication, Non repudiation, Robustness, Availability (Damiani, Anisetti, & Ardagna, Design and description of evidence-based certificates artifacts for services, 2011). Other proposed abstract security properties are: Deletion and Retention, Durability, Elasticity, Location, Continuity, Traceability, Anonymity, Personal Data Privacy, Accountability. It should however be noted that in some cases, properties could be conflicting, for example Anonymity (e.g., identity of the user who performed an action is not recorded) and Accountability (e.g., knowing the identity of the user who has performed a given action). In CUMULUS project a vocabulary of Security Properties is formally defined in another deliverable of Work Package 2 (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013)where a fine-grained security properties model is described in a multi-layered way. This vocabulary will be in use during the project.

In more general terms, for every Security Property we distinguish the attributes qualifying the property itself (e.g., assurance level and its measurement unit) and the attributes related to how the *Evidence* is collected (e.g., sample size, frequency, etc.). The formers are attributes of the Property because they define how the property value is represented along with its associated metric parameters, through performance and parametric attributes (see CUMULUS deliverable D2-1 (CSA, 2013)). The latters represent parameters concerning assurance, mechanisms and must be included in *Assertion* or *Certification Model* entities[1].

As a guide to modellers we can indicate that Security Properties can have two kinds of attributes, the first set (performance attributes) specify the Security Property definition itself, often summarized into a single value (a Boolean or a number), and the second set (parametric attributes) specify Security Property measurements, while some security property attributes are used to parametrize Assertions and finally Certification Models have attribute elements specifying evidence nature and location.

As an example in a Dynamic Testing modelling we may consider the AIS:integrity:data-alteration-detection Security Property, see definition in CUMULUS deliverable D2-1 (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013).This property contains a single "performance attribute" called data-alteration-detection and defined as Boolean, that measures the property itself, and no parametric attributes. Additional attributes can also be defined in Evidence (and in turn evidence collector and evidence aggregator) by means of assertion relation specifying all those elements useful for configuration and execution of the evidence collection process (e.g., Traffic threshold, Time interval between test execution). Finally the Dynamic Testing Certification Model could have attributes as Generation model, Number of test cases, Category that specify evidence nature and location. In summary our example of dynamic test model includes the Security Property [0xEEAF]:[AIS:integrity:data-alteration-detection] with attribute {data-alteration-detection = NO}, the Testing Evidence Instance [0xEFAB]:[Testing Evidence] with attributes { Traffic threshold = 200rps}; the Certification Model Instance [0xAEDF]:[DynamicTesting] with attributes {Generation model = URI; Number of test cases = 100; Category = Input Partitioning}.

The last entity of the module is the Commitment entity: it is a way to formulate a restriction over a data type of a Property. Commitment has been added to the Meta-Model in order to create a bridge to future

[1] We note that in some cases, properties in D2-1 will be extended to cover specific aspects of the cloud (e.g., multi-layer) and to exercise specific aspects of test-based and model-based certification.

security Service Level Agreements (SLAs) (Damiani, Ardagna, & Bezzi, Cluster Workshop on "Security Contracts", 2012).

All security-related *Commitments* should be part of SLA definitions. Our *Commitment* entity, however, could also be useful when querying the model. We note that *Commitments* are not part of certificates.

2.4. Certificate module

This section corresponds to section 3.4 in D2.2. There are no major amendments to this part, Certificate has now a LifeCycle.

The certificate module includes five entities: *Certificate*, *Assertion*, *Evidence*, *Context, Assumptions* and *Actor*.

A *Certificate* is a set of *Assertions* and is awarded by a Certification Authority for a specific *Target of Certification* (ToC*)*. Each *Certificate* has a *Life Cycle* with different states. Both the *Target of Certification* and the *Life Cycle* are defined by the *Certification Model*. Certification types that will be taken into account in CUMULUS are Test-based, Monitoring-based, TC-based, Hybrid, Incremental and Multi-layer. It is important to point out once more that the *Certificate* meta-class presented in the Meta-Model will be used through double instantiation (at model and at instance level) to produce a certification artefact, that is, an instance representing the outcome of the certification production process, equipped with values.

We define an *Assertion* as a constraint on a given *Property,* supported by a set of *Evidence*, for a given *Target of Certification*, and we define *Evidence* as a set of artefacts supporting a given *Assertion* and in turn a *Security Property*. An *Assertion* is related to *Security Property* (one many relationship from *Security Property* to *Assertion*), and to *Evidence* (1:N relationship); in other words a single *Assertion* has one or more *Evidence* and a single *Security Property* can have more *Assertions*.

The *Target of certification* (*ToC*) is the entity that will be certified and it can be the service under certification (i.e. SaaS layer), the platform deploying services (PaaS layer), the infrastructure hosting platforms and services (IaaS layer) or any combination of the above (in the case of multi-layer certification). The process that creates a Certification Model will concretize the ToC in different steps, starting from the generic cloud layer (SaaS, PaaS, IaaS, or combinations of them). ToC and its concretization process are detailed in next Sections (see Section 3.6 for the certification process).

Certificates refers to ToC an is bind to a cloud end-point (i.e. service at SaaS level or other cloud layers like, application engines, Virtual Machine, virtual infrastructure, network fabrics) and stored in a Certification Repository. Each time a new service is created and certified a new record is added to the Repository; the record contains information for the certification (assertions, etc.) and the specific TOC, via the Certification Model entity, plus the actual life cycle Certificate state (Issued, Suspended, Expired, Revoked, etc.).

The Actor is whoever is taking responsibility for Assertions. In the model schema we may have multiple Actors (for example Certification Authorities and Labs) and also Stakeholders such as CUMULUS customers (customers searching certified resources), providers of services and platforms and cloud providers.

The last entities of the module are the Context and Assumptions: the Context is the description of the Assumptions under which the Assertion has been made (including for example the configuration of all the framework components to be used during the certification process). We support the use of a lightweight Context, which can be simply defined by attribute-value pairs (such as certificate issue date). However heavyweight context representation is also allowed, as context format and its properties are both defined at Model level: each modeller is free to choose where to store the context information (a Context Server) and a proper format.

2.5. Certification Model module

This section corresponds to section 3.5 in D2.2. There are no major amendments to this part, Context and Target of Certification are now linked..

The lower right area in Figure 1 defines a certification model module, which includes *Certification Model*, *Evidence Collection* and *Evidence Aggregator*, *Life Cycle* and *TOC*.

Each *Certification Model* includes all the elements needed for a given class of certification: Service, Platform, and Infrastructure (S/P/I). There are also various types of *Certification Models*: Static Test-based (also called Offline), Dynamic Test-based (also said Online or Runtime), Monitoring-based and TC-based. The latter is based on Trusted Computing (TC) modules also called Trusted Platform Module or TPM, for example the Common Criteria Client TPM 1.2 Protection Profile developed by the Trusted Computing Group (TrustedComputingGroup, 2011). See also Table 2.

Certification Model	**Functioning**
Static Test-based (also defined Offline) CM	Tested in a pre-production environment
Dynamic Test-based (also defined Online or Runtime) CM	Tested in a production environment. For example a service is invoked and is immediately checked. This includes also periodic test to validate the Evidence.
Monitoring-based CM	Continuous monitoring of the operation of cloud services. Evidence has been collected in the past and still collected on a regular basis.
TC-based CM	Static certifications with TC modules.

Table 2 – Certification Models

The *Evidence Collector*, the *Evidence Aggregator* and the *Metrics/Conditions* entities provide information about how *Evidence* is collected and aggregated, how metrics are computed and compared to thresholds, and which starting conditions for online tests must be set.

A further entity is the *Target of Certification* (TOC): it does not only identify the instance of the service to be certified but also an instance of every cloud stack layer, including definitions of components, connections, interfaces, etc. As a result, the TOC enables a consumer to determine whether the security properties satisfiy their requirement, thanks to a detailed description of the certification perimeter.

Services expose interfaces to consumers but the service internal dynamics are often not disclosed. This lack of transparency contributes to a lack of assurance even if certain properties are certified. Increasing the transparency of the service architecture can mitigate this drawback. Moreover, when considering possible use cases such as the generation of “self-signed" certificates by service providers, the importance of service descriptions that increase the transparency of the service architecture could become significant. It provides a means of improving the trustworthiness of a service and compensates the lack of assurance given by the absence of the involvement of a recognized certification authority. When a service consumer discovers a service and fetches its certificate, in order to provide the service consumer with enough information to allow the security status of the service to be compared with the consumer’s own specific security requirements, the certificate must include a *Target of Certification*.

Often, the TOC refers to a service instance end-point (e.g. URI of a RESTFUL API or a WSDL PortType) but, in more general terms, the TOC identifies the perimeter of certification thus it may refers to components instances at a given cloud layer (or combination of layers in the case of multi-layer ToC) of the cloud protocol stack. Figure 3 shows examples of certifiable stack components considering a single layer TOC.

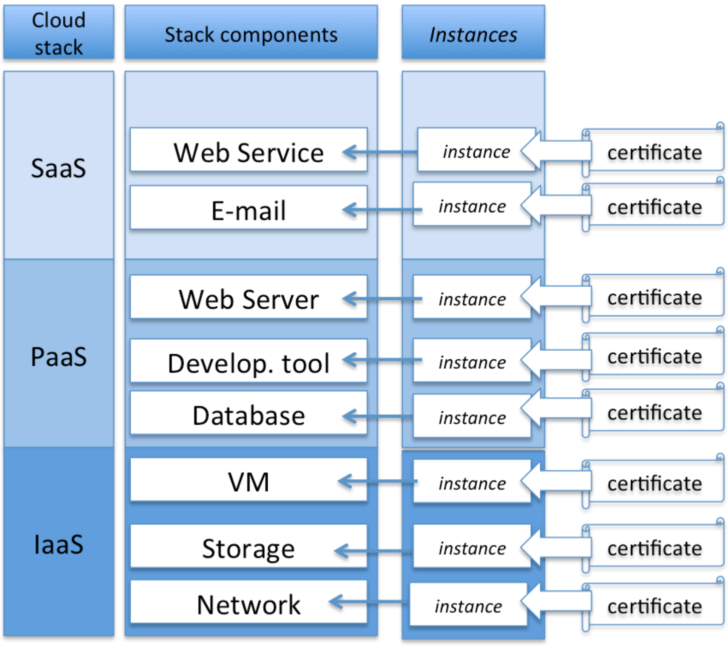


FIGURE 3 – CLOUD LAYERS, TOC, CLOUD COMPONENT INSTANCES AND CERTIFICATE BINDING (SINGLE LAYER)

The last entity of this module is the *Life Cycle*, which is linked to *Certification Model* and *Certificate*. When explaining the Certificate module we already stated that the *Certificate* meta-class would be used through double instantiation, at model and at instance level, to produce a certification artefact, i.e. an instance representing the outcome of the certification production process, equipped with values. The Meta-Model *Life Cycle* has been proposed with this artefact in mind; in other words, it is an artefact life cycle and not a certification process life cycle. Furthermore, this *Life Cycle* is intended as a guideline to individual certificate models, providing basic conceptual entities like revocation and validation. Particularly the *Certificate* might have different states in its life cycle: examples include *Valid*, *Invalid*, *Revoked*, *Renewed*, *Upgraded*, *Downgraded* (see Figure 4). At the model level, each certification type will define its own Certificate life cycle state activity diagram and individual modellers will redefine or develop their own life cycle with a different number of life cycle states.

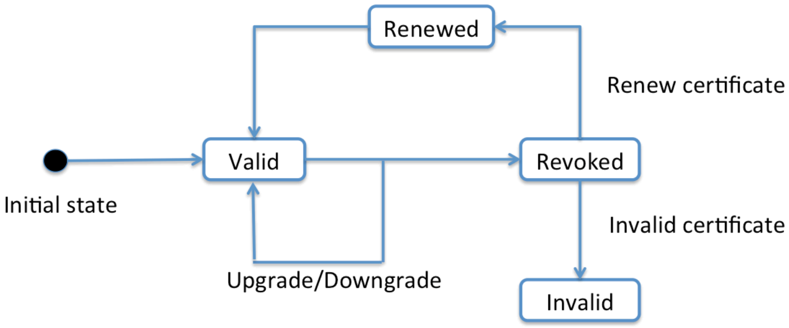


FIGURE 4 – EXAMPLE OF CERTIFICATE LIFE CYCLE

We also note that this specifies an implicit loose constraint between *Certification Model* and *Evidence* via the *Assertion* entity: the *Certification Model* can be used to decide which kind of *Evidence* an *Assertion* can support. In fact, in a specific *Certification Model*, say for example a Test-based model, all the *Evidence* that supports the *Assertion* must be also Test-based. In a hybrid certification scheme, there are two or more types of Evidence and each *Assertion* relies on a different *Certification Model*: when the *Certification Model* is test-based also the *Evidence* will be test-based, when the *Certification Model* is monitor-based the *Evidence* must be monitor-based. In other words, if *Assertions* have two related *Certification Models* then *Evidence* must be provided for both Models.

An example of this scenario is a *Security Property* whose *Assertions* are first certified with some preliminary Test-based *Evidence* and then subject to a Monitoring-based check on a later stage: in this case we have the same *Security Property*, that is proved by one *Certificate* including *Assertions* certified by the same Authority with two different *Certification models*.

We can also note that, as a consequence of what we said about the *Certification Models,* there are various kinds of *Evidence* model classes: Static Test, Dynamic Test, Monitoring and TC-based.

2.6. Certification Process module

This section has been fully developed with new concepts. Sub-section about chain of trust is also new.

The last module is the Certification Process module, which includes two entities: *Activity* and *Artefact (see Meta-Model in* Figure 1. The Certification Process models the activities of a very simple production process that an Actor, in our case a Certification Authority, could undertake in order to produce two different kinds of artefacts: the *Certification Models* and/or the *Certificates*.

An example of such a process in a Test-based scenario is a "definition process" that produces an XML-based *Certification Model* (XML-based CM) in different steps as described in the following of this section and in Figure 5.

The process starts by defining a XML schema (i.e., a .xsd file) that will be used to validate all the XML-based CMs produced during the process itself. Examples of xsd fragments are available in Sections 5 and 6.

Then a preliminary and generic XML-based CM, called *CM template*, is produced. It has a *Security Property* taken from a vocabulary (CSA, D2.1 DEVELOPMENT OF SECURITY PROPERTIES SPECIFICATION SCHEME AND SECURITY DEPENDENCY MODELS, 2013) and a generic *Target of Certification* (ToC), in our case just a single cloud layer (or a combination thereof) to be taken into account (SaaS/PaaS/IaaS layers).

Furthermore, the certification process refines the CM template including information on how and which tests must be performed, how the evidence must be produced, collected and archived (i.e., specifying generic and abstract test cases). This information is stored in a single element called *Collector* (see Sections 5 and 6). In this phase, a more specific Target of Certification is also specified, for example a generic database made by a vendor, a booking service, and the like.

Finally, starting from the refined CM template, a CM instance is produced where the Target of Certification is fully defined, for example the exact version of the database that has to be certified; in this final stage also the *Targets of Test* (ToTs), including specific information on the mechanisms tested by our certification process, are added to the XML of the Certification Model.

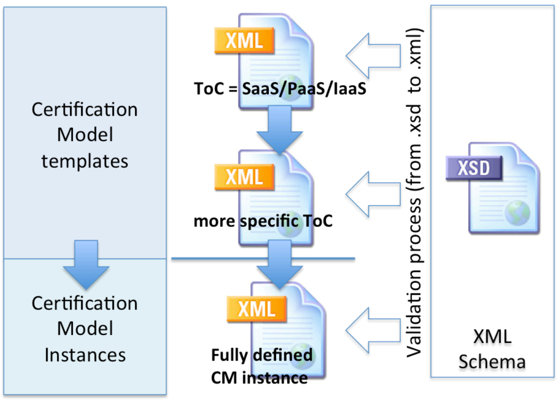

Certification Model templates
Certification Model Instances
XML
ToC = SaaS/PaaS/IaaS
XML
more specific ToC
XML
Fully defined CM instance
Validation process (from .xsd to .xml)
XSD
XML Schema

FIGURE 5 – DEFINITION PROCESS: MODELS AND INSTANCES

2.6.1. Chain of Trust

The concept of trust influences our life and has been debated much before the advent of IT technologies. With the success of IT technologies in general, and distributed systems in particular, the role of trust has gained increasing importance, becoming a fundamental building block for their success.

In the traditional software purchasing trust model, a software purchase involves two parties, a supplier *sp* and a customer *c*. In the following, we denote with $A_{en,ws}$ assertions made by an entity *en* over a system/service *ws*, and with $E_{en,ws}$ the evidence produced by an entity *en* over *ws* and supporting $A_{en,ws}$. Suppliers can make claims, that is, publish assertions about their software systems' functionalities, as well as on their non-functional properties. The customer *c*'s trust in an assertion $A_{sp,ws}$ made by the supplier *sp* is denoted $Tr(c, A_{sp,ws})$, where *Tr* takes discrete values on an ordinal scale (for example, for a Common Criteria certified product, one could take for *Tr* the relevant assurance level (EAL) value (1-7)). Traditionally, these claims were not supported by formal evidence on their truthfulness; rather trust in these claims was grounded on supplier reputation, especially in distributed systems. Certification approaches have then been adopted to increase trust in distributed systems (M. Anisetti, 2013).

Certification modifies *c*'s trust in assertions by introducing trusted external entities (Certification Authority - CA) in charge of collecting, validating, signing, and publishing assertions and related evidence. By doing so, a certification process also introduces new types of assertions that *c* can trust, describing the collection and validation processes. Traditional approaches to SOA certification assumes a chain of trust where the CA is available during the entire certification process and responsible for all activities necessary for certificate issuing (including signature). In this context, a certificate is issued at deployment time and eventually, renewed in case of modifications due to composition management or service versioning (M. Anisetti C. A., 2012).

A certification process in the cloud introduces the need of rethinking the trust model, the chain of trust, and their relation as part of the overall certification process. In fact, the assumptions made on the availability of the CA during the whole certification process cannot hold in a cloud scenario that is intrinsically dynamic and time dependent. In particular, a more complex chain of trust between the service provider, the customer, the certification authority, and the certification (Cumulus) framework must be defined with reference to the process described in Figure 5. This chain of trust clearly defines responsibilities for the different roles depending on the specific certification steps. This complexity is mainly due to the dynamics of the cloud environment, which requires online testing and monitoring after the issuance of a certificate for continuous support and management of certificates.

In this environment, differently from certification processes for SOA, we cannot assume a single signature by a trusted CA; rather, the signature process and eventually the responsibilities need to be spread across the certification process life cycle and the entities involved in the certification process. As a preliminary approach, the certification process life cycle can be decomposed in three different signing points where the information needed for building the Cumulus certificate is incrementally available: *i)* the generation of CM Template (offline), *ii)* the instantiation of CM Template in a CM Instance (or simply CM) for a given system (offline), *iii)* the generation of Cumulus Certificate based on the CM instance filled with all the details necessary for dynamic certification and lifecycle management (online). As discussed above, our approach cannot assume a single CA available for signing the artifacts produced during these three temporally subsequent steps. Within Cumulus, we are thus investigating different approaches to certification process management and certificate generation, with the corresponding chains of trust.

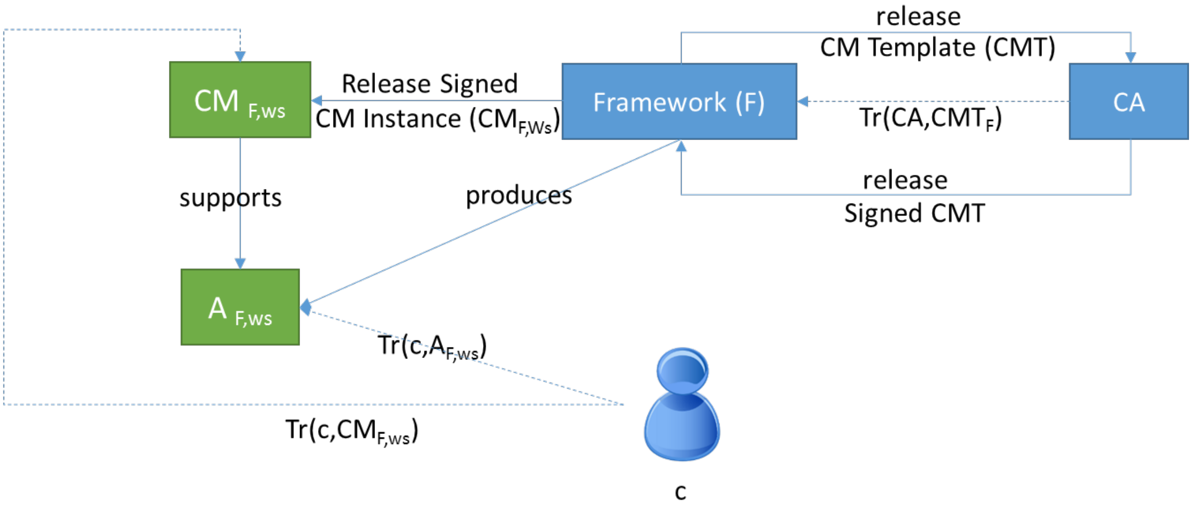


FIGURE 6 – COT (CHAIN OF TRUST)

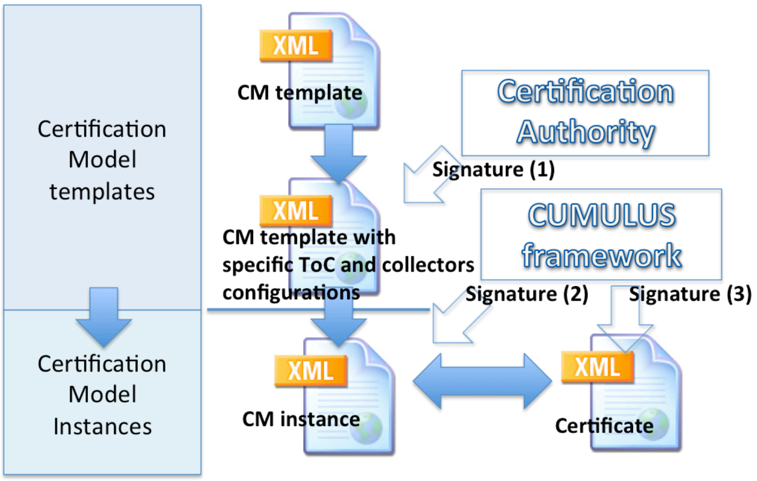


FIGURE 7 – SIGNATURE PROCESS

Our initial approach is a three-state certification process implementing a chain of trust based on three different signature processes (Figure 6 and Figure 7):

- *CM Template signature*: the CM Template is a partially specified CM (e.g., without the collector endpoints, the ToC elements), which describes the methodology for the certification (similarly to the protection profile of CC). The signature of the CM Template relies on the trust $Tr(CA,CMT_F)$ between the signing entity *CA* and the Cumulus Framework *F* managing the CM Template.
- *CM instance signature*: the Cumulus Framework is responsible to instantiate the CM Instance by filling all CM details, missing in the CM Template (collector configurations and test evidence *E*).[2] The signature of CM Instance at this step eventually relies on trust delegation, from the CA to the Cumulus Framework. It allows to support assertions on the security property of the given cloud system (*ws*) to certify $A_{F,ws}$, thanks to the evidence *E* generated by the instantiation of the CM Template.
- *Certificate signature*: this signature binds the CM to the certificate. The signed CM is filled with all the needed collectors and context configurations, and used to: i) execute real testing activities on the target of certification and, ii) produce the final security certificate.

Cumulus is investigating different approaches to certification process management, including certificate generation, along with corresponding chains of trust. Based on the preliminary analysis presented here, the mechanisms for obtaining the above three-state certification signature can be different starting from signature delegation, to incremental signature, or multiple signature protocols of partially filled documents. The latter consists of signing CM with empty collectors' endpoints URI and, only in a subsequent step, when they are available, sign their binding to the target of certification. The results of the study in progress will be reported in next relevant project deliverables.

[2] We note that the context and collectors' endpoints in the CM instance refers to laboratory endpoints, which will be then substituted in the final CM instance and certificate with real endpoints when the target of certification will be put in production.

3. CUMULUS Meta-Model compared with a Common Criteria Model

This section has no corresponding section in D2.2. It contains a comparison between the approaches to certification taken in CUMULUS and in Common Criteria. The comparison involves the CUMULUS meta-model and a suitable model for Common Criteria.

As already stated in Section 2, one of the objectives of defining a CUMULUS Meta-Model is to have a basic conceptualization of the main elements characterizing a security certification process. For this purpose it is also important to assess the alignment of the Meta-Model with already existing relevant approaches to security certification. Such a comparison may be especially useful for the validation of the CUMULUS Framework, where one of the significant dimensions selected is the *representation capability* (i.e., the capacity of representing pre-existing certification processes) (CUMULUS Consortium, D2.4 Final Certification Models, 2015).

A first comparison has been made with the Common Criteria (CC) approach, since CC is the security certification standard that is prevalent in Europe and world-wide. For this purpose, a model of the Common Criteria certification process (including evaluation) has been defined and its possible mapping to the CUMULUS Meta-Model has been analyzed with the objective of establishing to what extent the CC model could be seen as a test-based instance of the CUMULUS Meta-Model, so to have an indication of the representation capability of this (notice that this preliminary analysis did not aim at evaluating if/how CUMULUS may actually implement a CC certification process).

3.1. CC Model description

The elements of the CC model defined within CUMULUS and the relations between them are represented in the UML diagram in Figure below. For those elements representing concepts that have standard CC definitions, such definitions are reported with the respective references and possibly adapted. Definitions based on CC practice are given for the remaining elements, except *Life Cycle* that has been considered self-explanatory.

The element *Certification Criteria* represents all the rules that govern Common Criteria evaluation and certification, including:

- The Common Criteria themselves (Common Criteria v3.1 R4 Parts 1, 2 and 3, 2012);
- The associated evaluation methodology (CEM v3.1 R4, 2012);
- The additional guidelines produced by the Common Criteria Recognition Arrangement;
- The additional guidelines produced by single Certification Bodies.

A *Certification Body* (also called Evaluation Authority) sets the standards and monitors the quality of evaluations conducted by Evaluation Facilities within a specific community and implements the CC for that community by means of an administrative and regulatory framework called evaluation scheme (Common Criteria v3.1 R4 Part 1, 2012). A Certification Body completes the specification of Certification Criteria by providing its additional guidelines. It also accredits Evaluation Facilities and issues CC Certificates and Certification Reports.

An *Evaluation Facility* is an entity accredited by a Certification Body to act as an evaluator within a specific community. An Evaluation Facility evaluates TOEs and produces Evaluation Technical Reports.

A *CC Certificate* is an official document that attests the positive result of a certification process of a given TOE. A CC Certificate relies on Certification Criteria and depends on a Life Cycle that is defined by Certification Criteria.

A *Certification Report* (CR) summarizes the results of the certification process of a given TOE (Target of Evaluation). A Certification Report contains the relevant Evaluation Results and (a significant part of) Security Target plus other information, as the evaluated configuration of the TOE. A Certification Report relies on Certification Criteria and depends on a Life Cycle that is defined by Certification Criteria.

A *TOE* (Target Of Evaluation) is a set of software, firmware and/or hardware, possibly accompanied by guidance (Common Criteria v3.1 R4 Part 1, 2012). Every CC Certificate, Certification Report and Security Target is associated to one specific TOE.

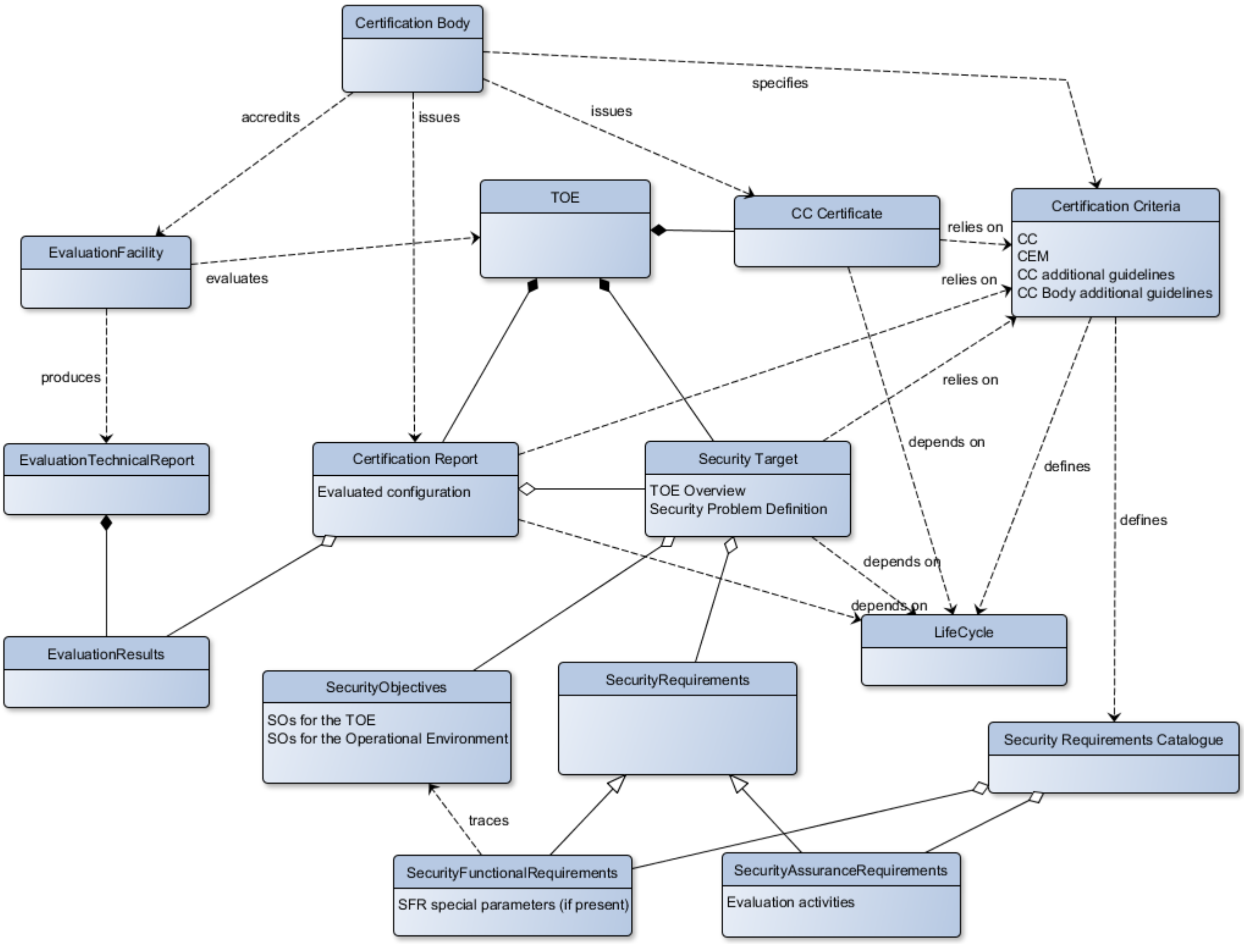


FIGURE 8 – COMMON CRITERIA MODEL

An *Evaluation Technical Report* (ETR) documents the overall verdict (i.e., the pass or fail statement issued by an Evaluation Facility with respect to the result of the evaluation of a given TOE) and its justification and is submitted to a Certification Body (Common Criteria v3.1 R4 Part 1, 2012). An Evaluation Technical Report basically contains Evaluation Results.

Evaluation Results are a summary of the results of all the evaluation activities performed during the evaluation process of a given TOE.

A *Security Target* (ST) is an implementation-dependent statement of security needs for a specific identified TOE (Common Criteria v3.1 R4 Part 1, 2012). A Security Target relies on Certification Criteria and depends on a Life Cycle that is defined by Certification Criteria. A Security Target contains Security Objectives and Security Requirements plus other information, including a TOE Overview and a Security Problem Definition (which is the description of the threats, Organizational Security Policies and assumptions on the Operational Environment that are addressed by the Security Objectives).

For simplicity, the relevant concept of Protection Profile (PP) is not represented in the CC model described above. A PP is an implementation-independent statement of security needs for a TOE type (Common Criteria v3.1 R4 Part 1, 2012). In practice, a PP is intended as a template for any ST of a TOE of the considered type. If a ST conforms to a PP, it explicitly claims this conformance and may also refer to the PP to complete its contents.

Security Objectives are informal definitions of security measures. They include Security Objectives for the TOE and Security Objectives for the Operational Environment (OE), which is the environment where the TOE is operated. A Security Objective for the TOE is an informal definition of some IT security measure that has to be implemented by the TOE to counter identified threats and/or satisfy identified Organisational Security Policies (OSPs). A Security Objective is implemented by the TOE by means of a set of Security Functional Requirements. A Security Objective for the OE is an informal definition of some IT or non-IT security measure that has to be implemented by the OE to counter identified threats and/or satisfy identified assumptions (on the OE) and/or OSPs (Common Criteria v3.1 R4 Part 1, 2012).

Security Requirements are either Security Functional Requirements or Security Assurance Requirements.

Security Functional Requirements (SFRs) are specifications (at a lower level with respect to Security Objectives) of some IT security measure (Common Criteria v3.1 R4 Part 1, 2012). SFRs trace to Security Objectives and may contain special parameters (details on SFR special parameters are given in Section 3.2).

Security Assurance Requirements (SARs) are specifications of some evaluation activities that concur in providing assurance that the TOE meets the SFRs claimed in the ST (Common Criteria v3.1 R4 Part 1, 2012). Notice that some SARs specifically address the development environment of the TOE.

The SFRs and SARs contained by a ST may be selected from a *Security Requirements Catalogue* that is defined by Certification Criteria.

3.2. CC Model mapping onto CUMULUS Meta-Model

The model of the Common Criteria certification approach described in Section 3.1 has been analysed to map its elements onto classes of the CUMULUS Meta-Model. The objective was not really to establish a rigorous mapping, but indeed to perform a summary assessment of the Meta-Model against a real world certification process.

In this sense, the mapping shown in Table 3 below should be read as a first attempt[3] to verify that each pivotal concept of the CC model could be seen as one (test-based) instance of some Meta-Model class (or a part of it). Each row of the Table 3 is organised as follows:, the first column reports one or more elements of the CC model; the second column reports the Meta-Model class(es) which the given elements could be mapped onto; finally, the third column reports some notes which provide additional information about how the mapping should be interpreted. The sign "+" is used to group elements when more elements of the CC model seem to be needed to instantiate one class of the Meta-Model or, vice versa, more classes of the Meta-Model seem to be instantiated by one element of the CC model. The expression "and" is used if more elements of the CC model seem to provide different instances of one class of the Meta-Model.

Some elements of the CC model do not appear in the Table 3 below but are covered by the mapping in an indirect way as follows:

- Evaluation Results are contained in ETR, which is covered by the mapping;

[3] This preliminary version of the mapping will be further refined in next deliverables, such as D6.5 Initial Validation Report.

- Security Assurance Requirements basically contain evaluation activities, which are covered by the mapping;
- Security Requirements and Security Requirements Catalogue contain SFRs and SARs, which are covered by the mapping (SARs are covered indirectly via evaluation activities, see above).

CC Model elements	**Meta-Model classes**	**Notes on the mapping**
TOE	TOC	Since the TOC concept identifies an instance of every cloud stack layer (see Section 2.5), it seems that, depending on TOE boundaries, TOC could map onto a single TOE or onto several TOEs of different certification processes
Certification Report + Security Target* + CC certificate	Certificate (instance of Artefact)	The CR, ST and CC Certificate (see section 3.1 for definitions) all provide a piece of information about the certification of the TOE which could be mapped onto the Meta-Model Certificate class. As a matter of fact, the Certificate is a container hosting: • The description of the security characteristics of a TOC (similarly to the ST); • The results of the evaluation/certification activities performed (similarly to the CR); • An official pronouncement of a Certification Authority about the above two points (similarly to the CC certificate). *The fact that the ST could have a reference to a PP is not relevant here (see Section 3.1).
Evaluation Technical Report	Evidence (part of Assertion)	The ETR seems to instantiate only a subset of the Evidence, because it usually does not contain the complete description of the tests performed during the CC evaluation. Since in Section 2.4 the Assertion is defined as a restriction on a Security Property supported by a set of Evidence, the association ETR-Evidence could be extended to Assertion. Notice that the ETR is usually not publicly available (only a subset of the information given in the ETR are publicly available in the Certification Report)
Life Cycle	Life Cycle	In CC there is not a well-defined lifecycle for CR, ST* and CC certificate (apart from guidelines for assurance continuity). This is in agreement with the fact that CUMULUS Meta-Model does not constrain a certification model to have a specific form of life cycle. * If the ST refers to a PP, then CR/ST/CC Certificate Life Cycle could need to refer to this PP
Certification Criteria	Certification Model	The mapping should be intended for Certification Criteria once they are applied to a given service for a given security property.

CC Model elements	Meta-Model classes	Notes on the mapping
		The corresponding (test-based) Certification Model could not provide a complete representation of the specified Certification Criteria (at least, it could represent CC evaluation activities only for the part which could be automated)
Evaluation activities (included in SARs)	Evidence Collector + Evidence Aggregator + Metrics/Conditions	Each of the Meta-Model classes Evidence Collector, Evidence Aggregator and Metrics/Conditions (see section 2.5 for definitions) could be instantiated by some part of the CC evaluation activities. Anyway, Meta-Model classes could not provide a complete representation of CC evaluation activities, since these are not meant to be automatic. Furthermore, the operations to be performed onto the TOE during evaluation activities are not specified exactly by CC. Anyway, this should not be a problem since the Meta-Model does not constrain a specific level of detail for the relevant classes (Evidence Collector, Evidence Aggregator, Metrics/Conditions)
Security Objectives for the OE (included in Security Objectives)	Context (description of the Assumptions)	Also some other portions of the ST (e.g., Non-TOE HW/SW/FW in the TOE Overview) and/or CR (TOE evaluated configuration) could be mapped onto the Context class of the Meta-Model. Since Context describes Assumptions, this association could be (directly) extended to Assumptions
Evaluation Facility and Certification Body	Actor (responsible for Activity)	Evaluation Facility is responsible for TOE evaluation/ETR production (Activity) Certification Body is responsible for CC Certificate/Certification Report Issue (Activity)
Security Objectives for the TOE (included in Security Objectives) + SFRs	Security Property	A precise mapping of only one element of the CC Model (i.e., either SOs for the TOE or SFRs) onto the Security Properties defined in CUMULUS seems to be not achievable because: • SOs are free text in natural language (unlike CUMULUS Security Properties); • SFRs in CC catalogue are not directly connected to abstract security properties. The mapping proposed apparently holds for associations with practical value in CUMULUS (i.e., for security properties similar to the ones in D2.1). For associations with less practical value in CUMULUS (i.e., for security properties defined to meet CC reality) an alternative mapping could be considered, where the Security Property in the Meta-Model is mapped to the collection of SFRs in a given ST. In this view, the CC SFR

CC Model elements	Meta-Model classes	Notes on the mapping
		catalogue would correspond to the reference set for the instantiation of a security property. Notice also that a ST can contain SFRs which are not in the CC catalogue, even though this is not recommended
SFR special parameters	Attribute (part of Security Property)	SFR special parameters may be used to specify mechanisms in a given SFR, or to select from a set of SFRs the one that corresponds to a given amount of functionalities. They are generally related to the level of assurance required to the evaluation

Table 3 – Certification Models Mapping between Common Criteria Model elements and CUMULUS Meta-Model classes

From the analysis conducted, it seems that the most significant CC concepts could be traced back onto the CUMULUS Meta-Model. Anyway, not all the classes of the Meta-Model are covered, since some of them (e.g., Commitment and Attribute Type) apparently may not be mapped to any CC concept. Moreover, the mapping is neither complete nor straightforward. In particular, as pointed out by several notes in the table, in general a CC concept does not exactly match with one Meta-Model class. The difficulties encountered are mainly due to the fact that CC have a human-oriented nature while the CUMULUS Meta-Model is designed to be suitable for (at least partially) automated security certification processes. However the identification of a complete and straightforward mapping between CC model and CUMULUS Meta-Model is out of the scope of this deliverable. On the other hand, the comparison between the CUMULUS Model and the given CC model highlighted the fact that the CUMULUS Meta-Model is able to capture the most significant elements of a real world certification process, thus providing a first answer about the representation capability of the Meta-Model itself.

4. Test-based Certification Model

This section corresponds to section 4 in D2.2. There are some major amendments: Collector elements have been fully developed (i.e. section 4.1.6 has been rewritten), the concepts of Abstrct Collectors and of Target of Test have been introduced. There are also some minor amendments in the terminology: Tested Security Property has been renamed as Security Property and Tester has been renamed as Signature in order to underline similarities with the Monitoring-based Certification Model.

CUMULUS aims at defining a security certification scheme that can be used to make trusted assurance information available in a cloud ecosystem; this is a recent idea that can allow users to evaluate different services and select the appropriate ones for service composition on the basis of their security needs. In other CUMULUS deliverable, i.e. D3-1 (CITY University, 2013) and D3-2 (University of Milan, 2014), we outline this process. A first goal is the definition of a certification infrastructure and of a schema for the specification and management of Static or Dynamic Test-based security certificates.

These kinds of certificates are evidence-based proofs that a test carried out on the software has given a certain result, which in turn shows that a given property holds for that software. The certificates are awarded by a *Certification Authority* for a specific *Target of Certification* and they aim to i) prove that some (static or dynamic) test-related assurance activities have been carried out, and ii) specify which security property these activities were meant to support. Checking CUMULUS certificates will allow consumers to ascertain that the assurance level provided by a certificate complies with its own requirements and will increase consumers' confidence that their security requirements can be met, or, better, that they have been met at the time and in the context of certification (Static or Offline Test-based certification process made in a pre-production environment) or when some conditions are met (Dynamic or Online Test-based certification process in a production environment).

Certification is carried out collaboratively by three main parties: i) a service provider that wants to certify its services; ii) a certification authority managing the overall certification process; and iii) a Lab accredited by the certification authority that carries out the property evaluation; in the following, we may use the term certification authority to refer both to the certification authority and its accredited Labs.

The Test-based Certification Model and Test-based Certificate must fully comply with the CUMULUS Meta-Model; the XML Schemas presented in this Section show the fundamental artefacts of Offline (Static) and Online (Dynamic) Test-based Certification process.

4.1. Test-based Certification Model XML Schema Description

This section corresponds to section 4.1 in D2.2. Some amendments have been made (the concept of Abstract Collector has been introduced, the Target of Certification includes now Targets of Tests).

The Test-based Certification Model fully complies with the CUMULUS Meta-Model. The XML Schema here presented shows the fundamental artefacts deriving from the model for Static/Offline and Dynamic/Online Test-based Certification. Then a XML incarnation of the Certification Model is provided.

The Certification Model specification schema includes the elements of the Figure 9 while the relations between Meta-Model classes and the models (Test-based Certification Model and Test-based Certificate) are shown in Table 4.

Model classes	Meta-Model classes
Collectors	Collector
Aggregator (sub element of the Collector)	Aggregator and Metrics & Conditions
LifeCycle	LifeCycle
ToC (Target of Certification) and Tots (Target of Tests)	Target of Certification
SecurityProperty (according to CSA definition in deliverable D2.1)	Security Property
Assertion (in the Certificate only)	Assertion
Signature	Actor
Context	Context
Tests	Evidence

Table 4 – Mapping between Model classes and Meta-Model classes for the Test-based Certification Model.

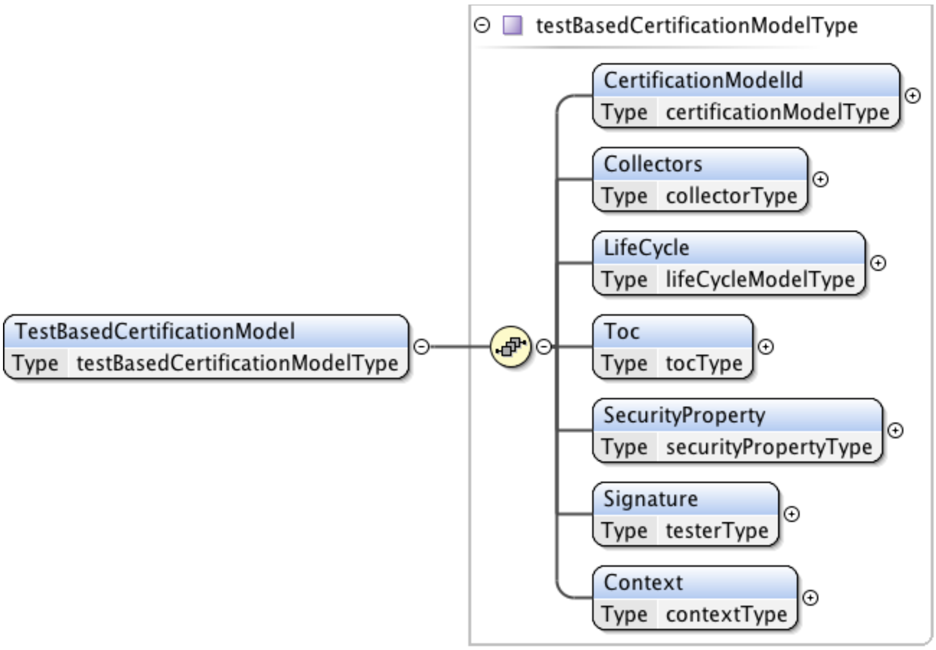


FIGURE 9 – TEST-BASED CERTIFICATION MODEL

A XML schema rendering of the description above is defined as follows:

```
<xs:element name="TestBasedCertificationModel" type="testBasedCertificationModelType"/>
          [...]
```

```
<xs:complexType name="testBasedCertificationModelType">
   <xs:sequence maxOccurs="1">
      <xs:element name="CertificationModelId" type="certificationModelType" minOccurs="1"/>
      <xs:element name="Collectors" type="collectorType"> </xs:element>
      <xs:element name="LifeCycle" type="lifeCycleType"/>
      <xs:element name="Toc" type="tocType" maxOccurs="1"/>
      <xs:element name="SecurityProperty" type="securityPropertyType"/>
      <xs:element name="Signature" type="signatureType"/>
      <xs:element name="Context" type="contextType"/>
   </xs:sequence>
</xs:complexType>
         [...]
```

The purpose of each Certification Model element is detailed in the next Sections.

All the Test-based Certification Models we define share the same structural elements of the schema above. The process of building a Certification Model has been already defined in Section2.6: it starts with a generic definition of the Certification Model where only the Security Property and the Collectors have been properly defined. Then a more specific Target of Certification (ToC) is added and, as final stage of the process, also the Targets of Test (ToTs) are added. The outcome is the XML code of the Certification Model instance.

4.1.1. Certification Model Id

The Certification Model Id was also in D2.2 version but without a dedicated Section. We use a dedicated Section to underline similarities between different CMs.

The CertificationModelID element represents a unique identifier of the CM instance.

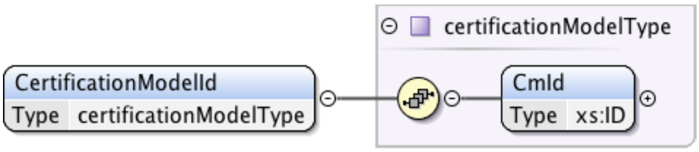


FIGURE 10 – TEST-BASED CM: CM ID ELEMENT TYPE

A XML schema that can be used for the XML code validation is the following.

```
<xs:element name="CertificationModelId" type="certificationModelType" minOccurs="1"/>
[...]
   <xs:complexType name="certificationModelType">
      <xs:sequence>
         <xs:element name="CmId" type="xs:ID"/>
      </xs:sequence>
   </xs:complexType>
```

Every XML incarnation is validated against the schema above.
in the XML code below a correct and concrete Id has been instantiated by CUMULUS framework.

```
   <CertificationModelId>
      <CmId>TEST000027</CmId>
   </CertificationModelId>
```

4.1.2. Life Cycle

> *This section corresponds to section 4.1.2 in D2.2. There are no amendments to this part of the test based certification model.*

The LifeCycle element in the Certification Model schema defines all the states and transitions of the Certificate.

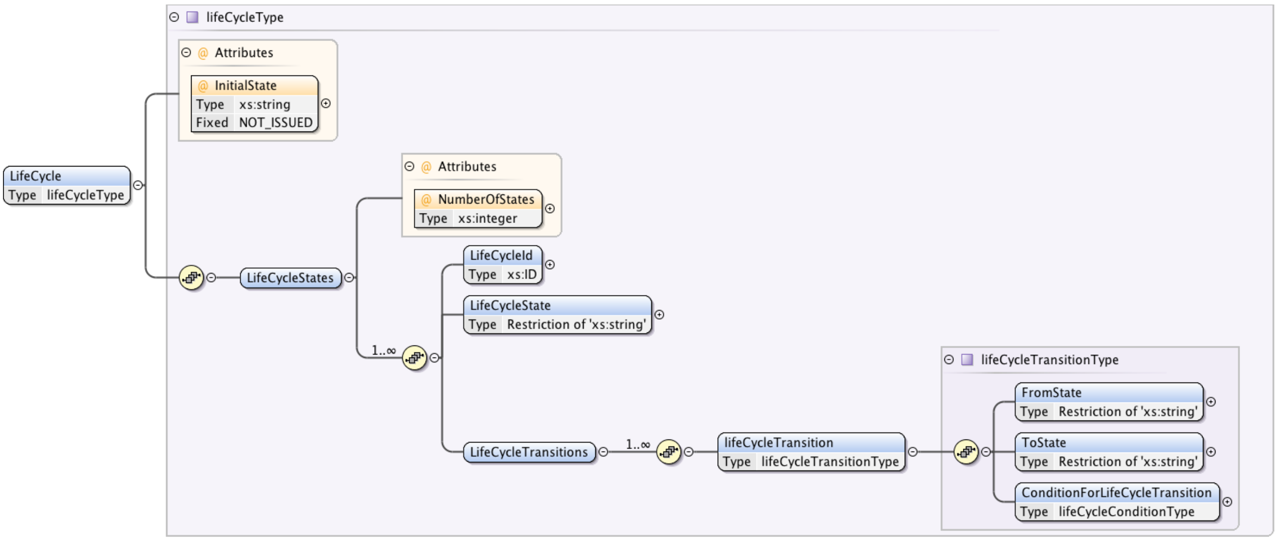


FIGURE 11 – TEST-BASED CM: LIFE CYCLE ELEMENT TYPE

In a typical Test-based Life Cycle scenario we take into consideration four states: ISSUED, SUSPENDED, EXPIRED and REVOKED. An initialState attribute in our schema is indicated as starting point of the full process, in our case it will be is set as NOT_ISSUED for all the Test-based Certification Models; when evidence is considered sufficient the certificate is created and issued with a certain validity period. In case the tests are not fully successful the certificate keeps the NOT_ISSUED state till it is revoked.

From the ISSUED state the Certificate can go either to an EXPIRED state, if the validity period has expired, or to REVOKED state, when evidence is contradictory, or to a SUSPENDED state, if for any reason the evidence is not considered sufficient when a dynamic test is made during the validity period of the certificate. If a Certificate is in this SUSPENDED state and the online tests are performed again with a fully positive outcome (i.e. the evidence is again considered sufficient), the Certificate goes back to the ISSUED state; see Figure 12; if instead the validity period has expired or dynamic tests produce contradictory evidence and the assertions become questionable, the certificate goes from SUSPENDED to respectively EXPIRED or REVOKED.

Once the certificate is REVOKED or EXPIRED, it cannot be validated any more and the full process has to be undertaken again in order to issue a new one.

A usual data flow in a Test-based scenario is:

- In pre-production offline tests are performed to issue a certificate with a validity period, i.e. the certificate has a ISSUED state; notice that in the Life Cycle there is no pending state where the certificate is issued but not valid yet.
- Then in production, the same tests are performed again (the so called online or dynamic phase): when the test outcome is positive the certificate is validated again (i.e. the ISSUED state is confirmed), when validity period expires the certificate goes to EXPIRED. If the test outcome is not positive, the certificate goes to REVOKED state, while if tests are not fully successful but test metrics are above a certain threshold the certificate can go to a temporary SUSPENDED state.

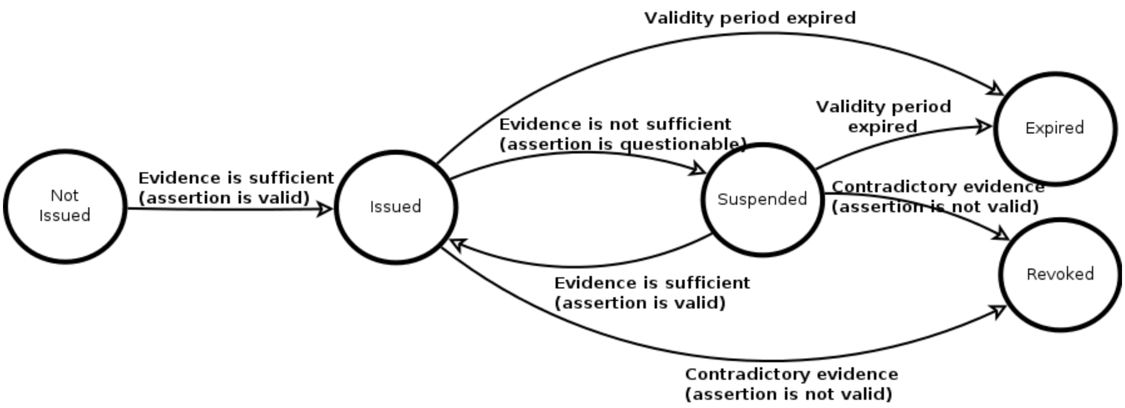

Validity period expired
Validity period expired
Expired
Evidence is not sufficient (assertion is questionable)
Not Issued
Evidence is sufficient (assertion is valid)
Issued
Suspended
Contradictory evidence (assertion is not valid)
Revoked
Evidence is sufficient (assertion is valid)
Contradictory evidence (assertion is not valid)

FIGURE 12 - TEST-BASED CM: LIFE CYCLE STATE TRANSITIONS

The Life Cycle described above is common to all possible Test-based Certification Model templates. For this reason, the XML schema of the generic Certification Model already includes the definition of the four states as previously defined, plus the initial state.

When a certificate is issued, a Validity Period is always attached to it. When the Validity Period expires, according to our actual Life Cycle, the full process must be start again, but this is not the only possible Life Cycle for a Test-based scenario: it is also possible to have another Life Cycle that allows confirmation of a certificate extending its validity period. In such a case after a successful online/Dynamic test the certificate is overwritten and Validity Period extended.

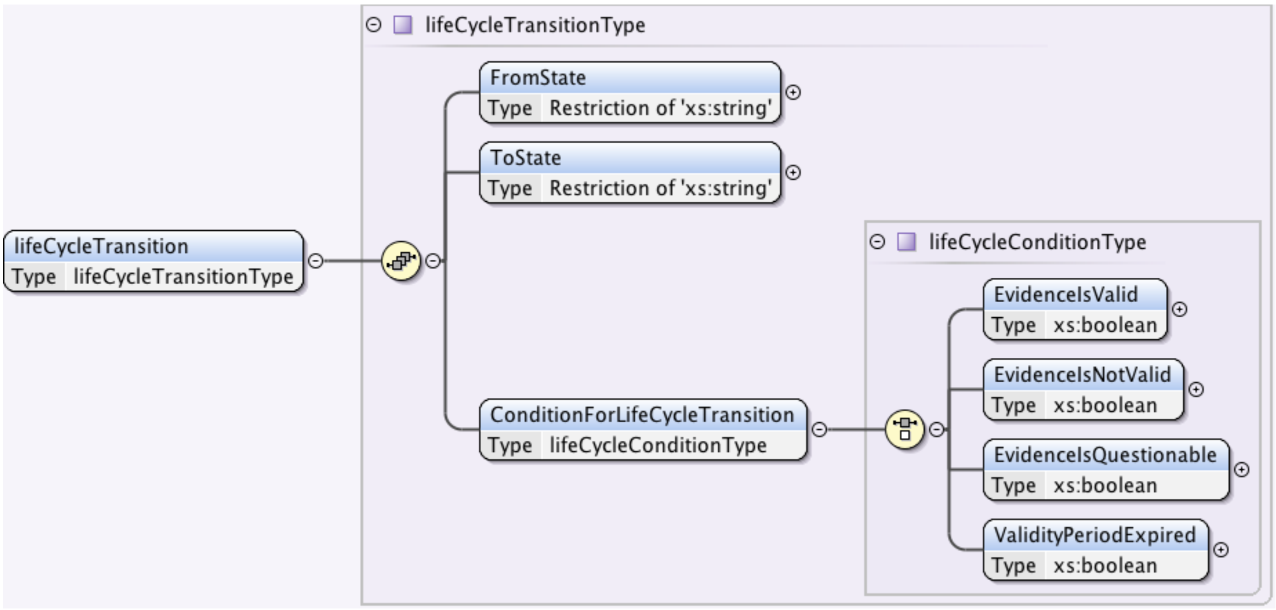


FIGURE 13 – TEST-BASED CM: LIFE CYCLE TRANSITION TYPE

Other Life Cycles are possible but in our current CUMULUS Test-based scenarios we remain tied to the process described above and the Life Cycle elements will be always be instantiated with these four states.

An excerpt for a Life Cycle XML schema is the following:

```
<xs:element name="LifeCycle" type="lifeCycleType"/>
[...]
    <xs:complexType name="lifeCycleType">
          <xs:sequence>
```

```
            <xs:element name="LifeCycleStates">
                <xs:complexType>
                    <xs:sequence maxOccurs="unbounded">
                        <xs:element name="LifeCycleId" type="xs:ID"/>
                        <xs:element name="LifeCycleState">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="NOT_ISSUED"/>
                                    <xs:enumeration value="ISSUED"/>
                                    <xs:enumeration value="SUSPENDED"/>
                                    <xs:enumeration value="REVOKED"/>
                                    <xs:enumeration value="EXPIRED"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="LifeCycleTransitions">
                            <xs:complexType>
                                <xs:sequence maxOccurs="unbounded">
                                <xs:element name="lifeCycleTransition"
                                type="lifeCycleTransitionType"> </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                    <xs:attribute name="NumberOfStates" use="required" type="xs:integer"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="InitialState" use="required" type="xs:string" fixed="NOT_ISSUED"/>
    </xs:complexType>
[...]
<xs:element name="lifeCycleTransition"
[...]
    <xs:complexType name="lifeCycleTransitionType">
        <xs:sequence maxOccurs="1">
            <xs:element name="FromState">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="NOT_ISSUED"/>
                        <xs:enumeration value="ISSUED"/>
                        <xs:enumeration value="SUSPENDED"/>
                        <xs:enumeration value="REVOKED"/>
                        <xs:enumeration value="EXPIRED"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ToState">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="NOT_ISSUED"/>
                        <xs:enumeration value="ISSUED"/>
                        <xs:enumeration value="SUSPENDED"/>
                        <xs:enumeration value="REVOKED"/>
                        <xs:enumeration value="EXPIRED"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ConditionForLifeCycleTransition" type="lifeCycleConditionType"/>
        </xs:sequence>
    </xs:complexType>
```

XML code is validated against the schema above. A XML example of a CM template is in the following box. As an example the NOT_ISSUED state is the first Life Cycle state to be described: it has only one transition allowed according to our Life Cycle diagram, from NOT_ISSUED to ISSUED when “evidence is valid”; there is no intermediate state where the certificate is created but not valid yet.

```
<LifeCycle InitialState="NOT_ISSUED">
  <LifeCycleStates NumberOfStates="5">
    <LifeCycleId>ID1</LifeCycleId>
    <LifeCycleState>NOT_ISSUED</LifeCycleState>
    <LifeCycleTransitions>
      <lifeCycleTransition>
        <FromState>NOT_ISSUED</FromState>  <ToState>ISSUED</ToState>
        <ConditionForLifeCycleTransition> <EvidenceIsValid>true</EvidenceIsValid>
        </ConditionForLifeCycleTransition>
      </lifeCycleTransition>
    </LifeCycleTransitions>
    <LifeCycleId>ID2</LifeCycleId>
    <LifeCycleState>ISSUED</LifeCycleState>
    <LifeCycleTransitions>
      <lifeCycleTransition>
        <FromState>ISSUED</FromState> <ToState>EXPIRED</ToState>
        <ConditionForLifeCycleTransition>  <ValidityPeriodExpired>true</ValidityPeriodExpired>
        </ConditionForLifeCycleTransition>
      </lifeCycleTransition>
      <lifeCycleTransition>
        <FromState>ISSUED</FromState>  <ToState>SUSPENDED</ToState>
         [...]
        <FromState>ISSUED</FromState>  <ToState>REVOKED</ToState>
         [...]
        <FromState>SUSPENDED</FromState>   <ToState>ISSUED</ToState>
         [...]
    <LifeCycleId>ID3</LifeCycleId>
    <LifeCycleState>SUSPENDED</LifeCycleState>
    <LifeCycleTransitions>
      <lifeCycleTransition>
        <FromState>SUSPENDED</FromState>  <ToState>EXPIRED</ToState>
         [...]
        <FromState>SUSPENDED</FromState>  <ToState>ISSUED</ToState>
         [...]
        <FromState>SUSPENDED</FromState> <ToState>REVOKED</ToState>
         [...]
        <FromState>ISSUED</FromState>  <ToState>SUSPENDED</ToState>
         [...]
    <LifeCycleId>ID4</LifeCycleId>
    <LifeCycleState>EXPIRED</LifeCycleState>
    <LifeCycleTransitions>
      <lifeCycleTransition>
        <FromState>ISSUED</FromState>  <ToState>EXPIRED</ToState>
         [...]
        <FromState>SUSPENDED</FromState>  <ToState>EXPIRED</ToState>
         [...]
    <LifeCycleId>ID5</LifeCycleId>
    <LifeCycleState>REVOKED</LifeCycleState>
    <LifeCycleTransitions>
         [...]
        </lifeCycleTransition>
    </LifeCycleTransitions>
  </LifeCycleStates>
</LifeCycle>
```

4.1.3. Target of Certification (ToC) Element

This section corresponds to section 4.1.3 in D2.2. The concept of Target of Tests, inside a Target of Certification, has been introduced.

The Target of Certification (ToC) element in the Certification Model schema is defined below.

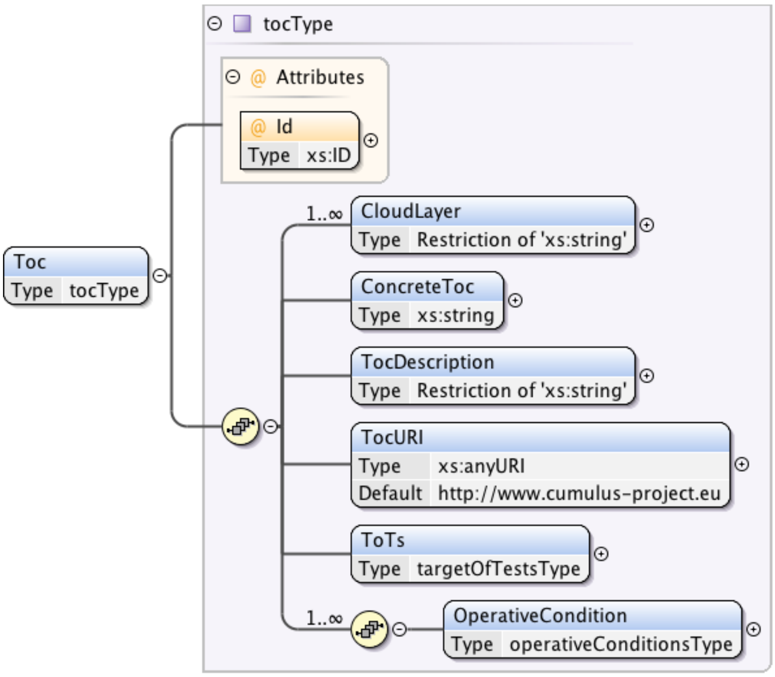


FIGURE 14 – TEST-BASED CM: TOC ELEMENT TYPE

A Certification Model defines a Target of Certification instance that is certified for the actual Security Property, and the ToC describes the perimeter of what we want to certify.

In the Certification Model the Target of Certification element has an Identifier attribute generated by the framework, in order to guarantee the alignment for Test-based and Monitoring-based CM instances.

The other sub-elements are: the generic layer in the cloud stack layer (i.e. SaaS, PaaS or IaaS; it can happen the two or three of these layers are specified, as for example SaaS + PaaS), in the ConcreteToc sub-element (i.e. the specific concrete instance to be certified) a textual description, a URI identifier (i.e. the reference to the services to be certificated), the Targets of Test (ToTs) sub-element (the sub-element specifies the accessible APIs for CUMULUS framework) and the operative condition sub-elements (they describe the operational conditions under which the ToC works and include all the necessary technical information: the vendor and the release related info, installation constraints, etc.)

An excerpt of the XMD schema is shown below.

```
<xs:element name="Toc" type="tocType" maxOccurs="1"/>
[...]
  <xs:complexType name="tocType">
    <xs:sequence>
      <xs:element name="CloudLayer" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="SaaS"/>
            <xs:enumeration value="PaaS"/>
            <xs:enumeration value="IaaS"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ConcreteToc" type="xs:string"> </xs:element>
      <xs:element name="TocDescription">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="WS"/>  <xs:enumeration value="Application"/>
            <xs:enumeration value="DBMS"/>  <xs:enumeration value="WEBSERVER"/>
            <xs:enumeration value="EMAIL"/>  <xs:enumeration value="CRM"/>
            <xs:enumeration value="SDK"/>   <xs:enumeration value="VIRTUALMACHINE"/>
            <xs:enumeration value="HD"/>   <xs:enumeration value="SWITCH"/>
            [....]
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="TocURI" type="xs:anyURI" default="http://www.cumulus-project.eu"/>
      <xs:element name="ToTs" type="targetOfTestsType"/>
      <xs:sequence maxOccurs="unbounded" minOccurs="1">
        <xs:element name="OperativeCondition" type="operativeConditionsType"> </xs:element>
      </xs:sequence>
    </xs:sequence>
    <xs:attribute name="Id" use="required" type="xs:ID"> </xs:attribute>
  </xs:complexType>
```

An example of a XML code is shown below.

```
    <Toc Id="ID001">
      <CloudLayer>SaaS</CloudLayer>
      <ConcreteToc>e-health v1.0</ConcreteToc>
      <TocDescription>Application</TocDescription>
      <TocURI>10.0.0.155</TocURI>
      <ToTs> [.....] </ToTs>
      <OperativeCondition>
        <TocTechnicalSpecifications>
          <TocVendor>ATOS</TocVendor>
          <TocRelease>1.0</TocRelease>
          <TocDate>2014-09-24</TocDate>
        </TocTechnicalSpecifications>
      </OperativeCondition>
    </Toc>
```

4.1.4. SecurityProperty Element

This section corresponds to section 4.1.4 in D2.2. A minor amendment was to rename TestedSecurityProperty as SecurityProperty to underline similarities between different CMs.

The SecurityProperty element in the schema defines the Security Property, which has to be certified by the Certification Model instance.

The *securityPropertyType* is an extension of the *propertyDefType*, already defined in (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013) and includes some other parameters relevant to the property definition.

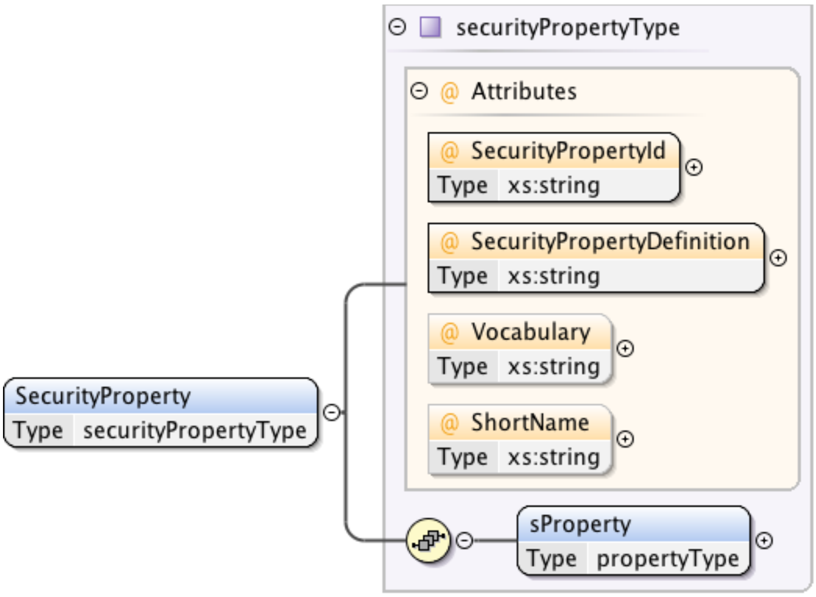


FIGURE 15 – TEST-BASED CM: SECURITY PROPERTY ELEMENT TYPE

The *propertyDefType* has Performance attributes and Parameters attributes to make the property concrete. The representation of *propertyDefType* is the following:

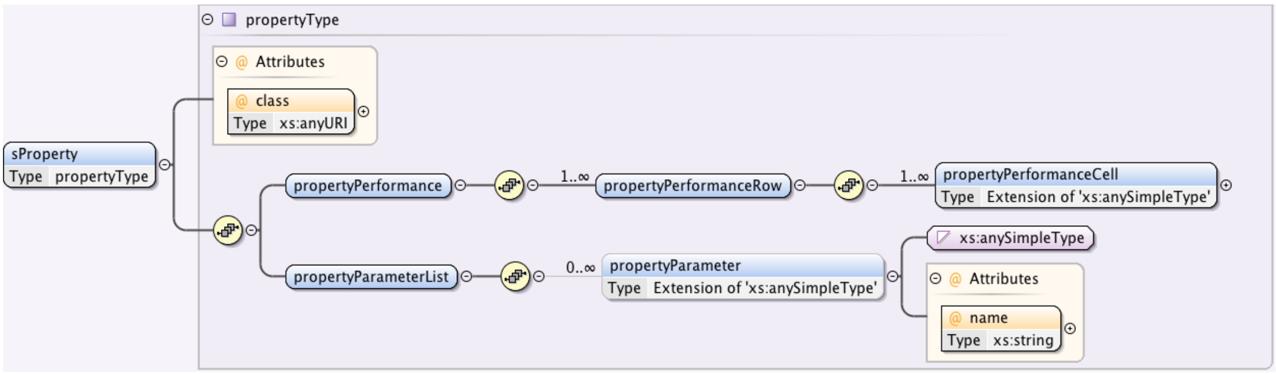


FIGURE 16 – TEST-BASED CM: SECURITY PROPERTY SUB-ELEMENT TYPE

The *class* attribute is the name of the property according to CSA vocabulary (for example: *AIS:confidentiality:service-provider-data-access-level* and *AIS:confidentiality:external-data-exchange-confidentiality*).

Notice that Assertions are included neither in the Security Property element nor in any other element of the Certification Model: in fact, when a Certification Model is created, an Actor (for example a Certification Authority that takes decisions about a Certification Model) does not know yet which kind of Assertions will be taken into consideration to certify the Property, and for this reason, a Certification Model instance does not contain any Assertion instance.

The XSD schema and XML code of some security properties used in our scenarios are shown in the following boxes.

The XML schema of the SecurityProperty element is the following:

```
<xs:element name="SecurityProperty" type="securityPropertyType"/>
[...]
  <xs:complexType name="securityPropertyType">
    <xs:sequence maxOccurs="1" minOccurs="1">
```

```
      <xs:element name="sProperty" type="propertyType"/>
    </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>
 [...]
      <xs:element name="propertyPerformance">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="propertyPerformanceRow" minOccurs="1"
              maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="propertyPerformanceCell" minOccurs="1"
                    maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:anySimpleType">
                         <xs:attribute name="name" type="xs:string"
                         use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
[...]
      <xs:element name="propertyParameterList">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="propertyParameter" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:anySimpleType">
                    <xs:attribute name="name" type="xs:string" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

The XML code of a Certification Model instance is the concretization of the XSD Certification Model and the XML code is verified against XMD Certification Model template.

Here is an example with the a property used in our ATOS scenarios: IAM:identity-assurance:user-authentication-and-identity-assurance-level. It requires a Performance attribute (see level in propertyPerformanceCell).

```
  <SecurityProperty SecurityPropertyId="Id101" SecurityPropertyDefinition="This property measures the strength of
the mechanism used to authenticate a user, on a scale from 0 to 4, notably taking into account identity proofing,
credential security during transfer and storage.">
```

```
      <sProperty class="http://cumulus-project.eu/security-properties#IAM:identity-assurance:user-authentication-and-identity-assurance-level">
        <propertyPerformance>
          <propertyPerformanceRow>
            <propertyPerformanceCell name="level">1</propertyPerformanceCell>
          </propertyPerformanceRow>
        </propertyPerformance>
        <propertyParameterList>
        </propertyParameterList>
      </sProperty>
    </SecurityProperty>
```

4.1.5. Signature Element

This section corresponds to section 4.1.6 in D2.2. The element was previously called Tester, but it hass been renamed Siganture to underline the common parts of different CMs.

The Actor in the Meta-Model is represented by the Tester whose signature is an element of the XML schema description: the Tester can be a Certification Authority dealing with the overall certification process or a Lab accredited by the Certification Authority.

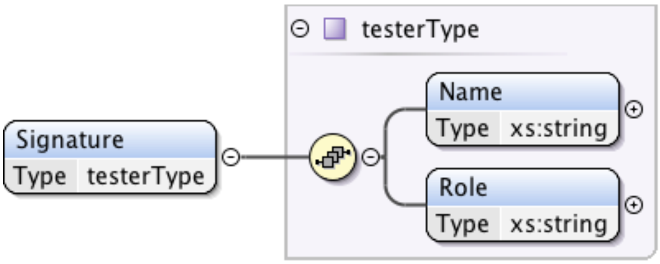


FIGURE 17 – TEST-BASED CM: SIGNATURE ELEMENT TYPE

An excerpt of the XMD schema is shown below.

```
<xs:element name="Signature" type="testerType"/>
[...]
     <xs:complexType name="testerType">
          <xs:sequence maxOccurs="1">
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Role" type="xs:string"/>
          </xs:sequence>
     </xs:complexType>
```

A XML fragments from a CM instance is:

```
    <Signature>
      <Name>SesarLab</Name>
      <Role>Laboratory</Role>
    </Signature>
```

4.1.6. Collectors Element

This section corresponds to section 4.1.1 in D2.2. The section has been rewritten and the concept of Abstract Collector has been introduced.

The Collectors element in the Certification Model schema is defined below. It includes all the sub-elements that perform this task: Abstract Collector, the Collector list and an EventBusCollector.

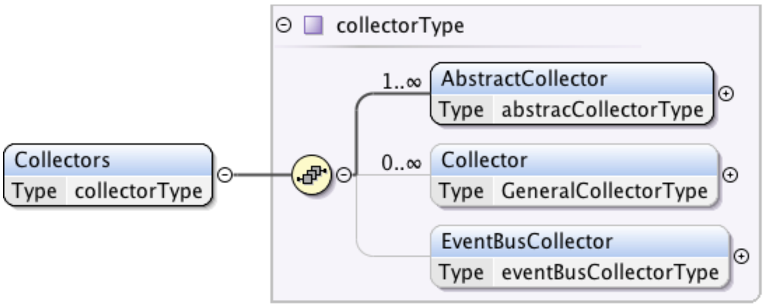


FIGURE 18 – TEST-BASED CM: COLLECTORS ELEMENT TYPE

In principle the Abstract Collector element in the XML schema defines the Collector in a very generic way, i.e. how tests must be performed, how the evidence must be produced, collected and archived in a Test-based certification scenario. See Figure below.

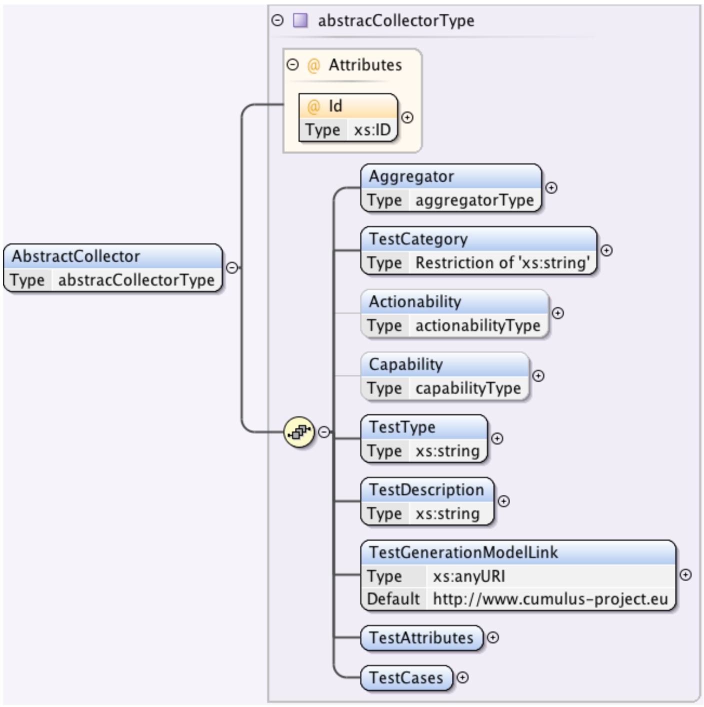


FIGURE 19 – TEST-BASED CM: ABSTRACT COLLECTOR SUB-ELEMENT

A Collector element in the XML schema defines instead how to activate the Abstract Collector in a real scenario. For this reason it refers to an Abstact Collector that has been described above. It has also attributes that describe the Static and Dynamic environments, a generic text escription, an expiration time. See Figure below.

As an example, in a Static/Offline Testing environment, when a testing client probe is assessing a specific Web Service (SaaS layer) and certifying “confidentiality in transit”, the collection process is the process of sending and receiving a data stream from the probe to the Web Service under test in a pre-production environment. In this case the isStatic attribute is set to “True” and the Collector specifies the inputs that have to be submitted and the output that has to be collected.

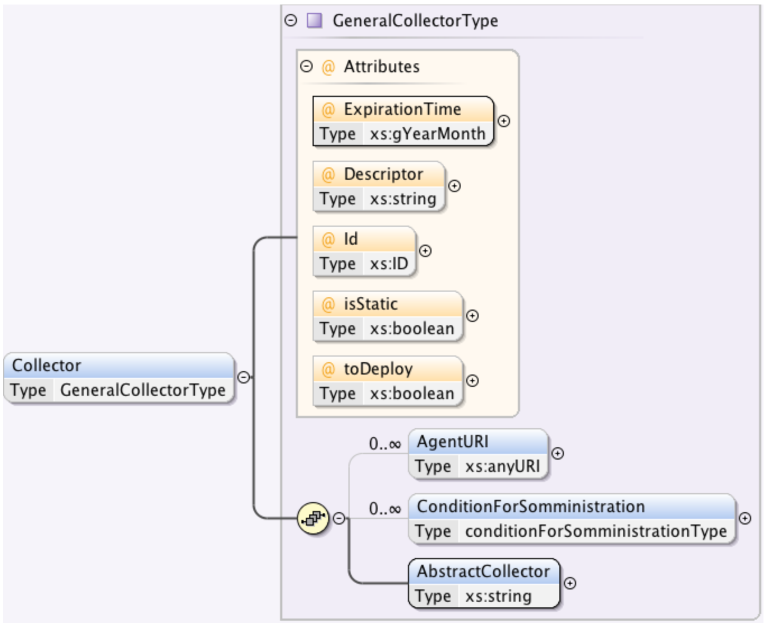


FIGURE 20 – TEST-BASED CM: COLLECTOR SUB-ELEMENT

Onthe other hand in a Dynamic Test-based certification, the client testing probe works either synchronously or asynchronously in a production environment. In the first case the probe waits for a feedback after its stimulus, in the second case the probe sends its input and then reads or receives parameters from a different channel, in particular the process could evaluate parameters of the infrastructure itself or waits from an asynchronous feedback from the server. For example a typical situation of the synchronous scenario is when tests are injected continuously with a delta time in between. In an asynchronous scenario tests are not injected continuously but only when a certain condition is met, for example when network traffic is below a certain threshold and tests do not affect the ordinary operations of the service.

Some of the ancillary parameters could also be read in an Event Bus (see EventBusCollector element in Figure below), which can provide generic platform events.

For example these events can be useful when evaluating a security property such as Confidentiality. In this case, events such as the access and utilisation of server side encryption libraries could provide valuable information to the Tester.

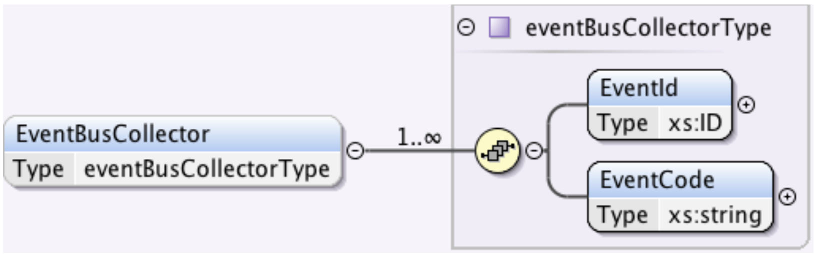


Figure 21 – Test-based CM: Event Bus Collector Sub-Element

An excerpt of the XMD schema of a Certification Model schema is shown below.

```
<xs:element name="Collector" type="collectorType"> </xs:element>
[...]
  <xs:complexType name="collectorType">
    <xs:sequence>
      <xs:element name="AbstractCollector" type="abstracCollectorType" maxOccurs="unbounded"
        minOccurs="1"/>
      <xs:element name="Collector" type="GeneralCollectorType" minOccurs="0"
        maxOccurs="unbounded"> </xs:element>
      <xs:element name="EventBusCollector" type="eventBusCollectorType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
[...]
```

The XML code for a Certification Model template is verified against the schema above and then a XML instance is produced.

```
   <Collectors>
        <AbstractCollector Id="abstract1">
          <Aggregator >
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <TestType>Model Control Flow</TestType>
          <TestDescription>Send user/pass (with malformed pass) to registerUser API. RegisterUser result must be
"fail".</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model005.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>1</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
          <TestCase>
          <ID>1</ID>
          <Description>user generation</Description>
           <TestInstance Operation="registerUser">
            <Preconditions/>
            <HiddenCommunications/>
            <Input>username=fred; password="12345"</Input>
            <ExpectedOutput>fail</ExpectedOutput>
            <PostConditions></PostConditions>
           </TestInstance>
           </TestCase>
          </TestCases>
        </AbstractCollector>
        <AbstractCollector Id="abastract2">  [...]   </AbstractCollector>
[...]
    <Collector ExpirationTime="2014-09" toDeploy="true" isStatic="true" Id="col1">
      <ConditionForSomministration><DeltaTime>12:00:00</DeltaTime></ConditionForSomministration>
      <AbstractCollector>abstract1</AbstractCollector>
    </Collector>
[...]
    <Collector ExpirationTime="2014-09"  toDeploy="true" isStatic="true" Id="col2">
      <ConditionForSomministration>
        <Event>
          <Action>Login</Action>
          <Condition> failed
          </Condition>
          <Value>100</Value>
        </Event>
      </ConditionForSomministration>
      <AbstractCollector>abstract4</AbstractCollector>
    </Collector>
  </Collectors>
```

4.1.7. Aggregator Sub-Element

This part was not present in D2.2 version, since it was included in the Collector Section (Section 4.1.1 in deliverable D2.2)

The Aggregator sub-element describes how to collect the test outcomes and how the evidence must be aggregated. In our Certification Model the Aggregator is a sub-element of the Abstract Collector.

This is also related to the Metrics and Conditions Class in the Meta Model. In fact this element deals also with criteria for interpreting test results in terms of sufficiency of collected results and includes performance thresholds that are appropriately scaled and arranged in different levels to provide different levels of assurance.

In a Test-based scenario Static/Offline tests are performed in pre-production and then saved. When test are performed again in production the results are compared with the evidence collected before in pre-production and must satisfy a metric condition, for example a threshold or a percentage. The thresholds could be of different types like test number, coverage of given test types, test duration/timing (e.g. one day of tests). In other circumstances the metric condition could be a simple Boolean.

For example when tests about Confidentiality security property are performed on a remote storage, a server agent on the cloud side could be able to provide whether the storage is properly encrypted providing a Boolean value as a feed back to a probe, and, as a consequence, can indicate whether Confidentiality security property can be certified or not. The conditions could also include rules to compare actual results with expected results, for example, when considering the Availability security property, the tests performed during one day of tests should have percentage-of-processed-requests of 95%. The conditions deal also with problem of how possible conflicts could be resolved, for example if failure of a limited number of tests affects the full validity of an already issued Certificate.

The Aggregator element in the Certification Model schema is defined below.

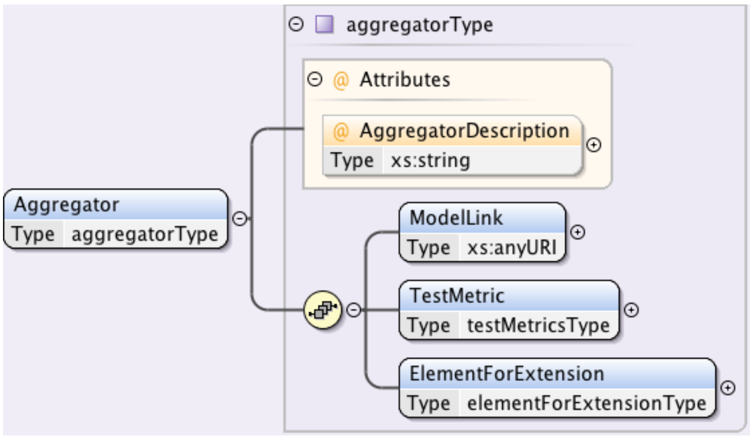


FIGURE 22 – TEST-BASED CM: AGGREGATOR SUB-ELEMENT

The Aggregator element in the specific Certification Models may have only a subset of these elements. The TestMetrics element contains the set of metrics used to evaluate the quality, completeness, and suitability of the tests executed on the ToC. All these metrics can be used in the matching between users' preferences and service certificates, and in the comparison between certificates to find the best that matches users' preferences. Examples of test metrics are *PathCoverage* and *InputCoverage*. We note that the metrics can be measured: i) using all available test cases thus providing a summary on the quality of the certification process (as assumed in this deliverable); ii) per test category, meaning that only test cases of a given category are considered; iii) for each pair of test category and test type, meaning that only test cases generated with a given pair are considered. The calculation of the metrics at different levels using different subsets of test cases permits to support enhanced scenarios of matching and comparison. To this aim, in future version of the certificate, we would need to define multiple elements TestMetrics each one referring to a given test category and test type.

An example of the XMD schema is shown below.

```
<xs:element name="Aggregator" type="aggregatorType"/>
[...]
   <xs:complexType name="aggregatorType">
      <xs:sequence>
         <xs:element name="ModelLink" type="xs:anyURI"/>
         <xs:element name="TestMetric" type="testMetricsType"/>
         <xs:element name="ElementForExtension" type="elementForExtensionType"/>
      </xs:sequence>
      <xs:attribute name="AggregatorDescription" type="xs:string"> </xs:attribute>
   </xs:complexType>
```

4.1.8. Context Element

This section corresponds to section 4.1.7 in D2.2. The element now focues on the configuration of the tools used in the test-based certification process.

The Context element details the configuration of tools that were used in the certification process.

The Context element in the Certification Model schema is defined below.

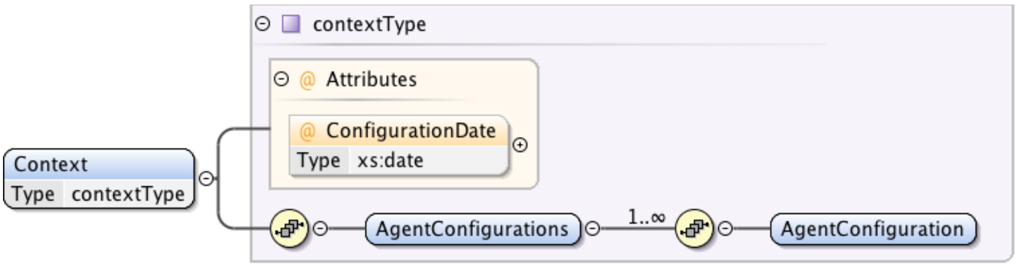


FIGURE 23 – TEST-BASED CM: CONTEXT ELEMENT TYPE

The sub-element called AgentConfiguration describes a cloud side “fully trusted” software initial configuration. This sub-element will be fully described in the next version of this Deliverable.

An excerpt of the XMD schema of a generic Certification Model is shown below.

```
<xs:element name="Context" type="contextType"/>
[...]
  <xs:complexType name="contextType">
    <xs:sequence>
      <xs:element name="AgentConfigurations">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="AgentConfiguration"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="ConfigurationDate" type="xs:date"/>
  </xs:complexType>
```

4.2. Test-based Certificate XML Schema

This section corresponds to section 4.2 in D2.2. There are no major amendments to this part.

The XML Schema presented below shows the fundamental elements for a Test-based certificate. In the Test-based certification process first a Certification Model instance is created and then passed to the testing module, which creates a Certificate with a NOT_ISSUED state and starts injection of tests. Once sufficient evidence has been collected the Certificate switches to an ISSUED state.

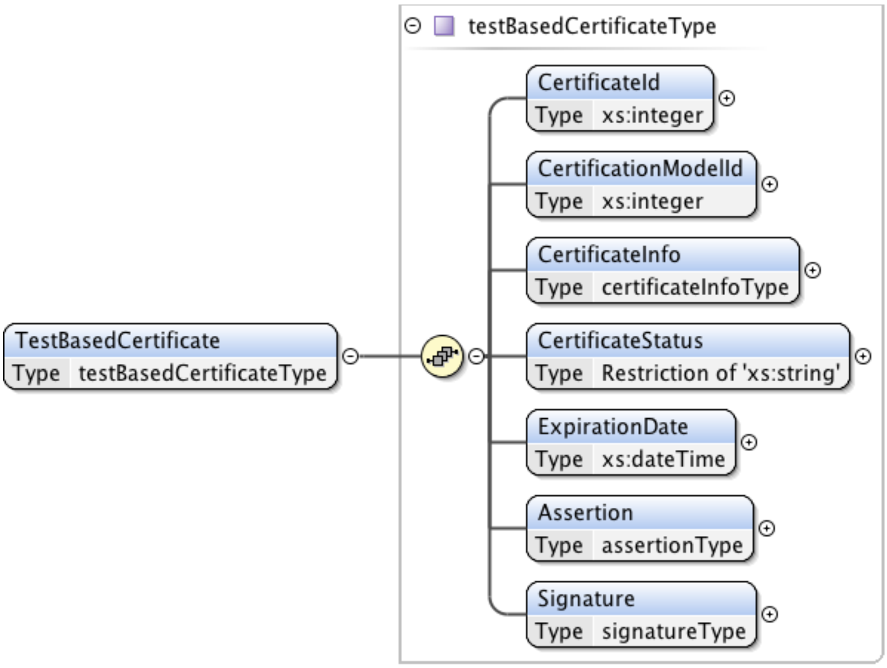


FIGURE 24 – TEST-BASED CERTIFICATE

The XML Schema includes the following elements:

- the CertificateId element is a unique identifier,
- the CertificationModelId element provides a reference to the certification model instance,
- the CertificateInfo element that provides the information about the type of certification (in our case it is always “Test-based”), the Date of Certification, the mode (i.e. whether the Certificate is Static/Offline or Dynamic/Online),
- the CertificateStatus element holds the current Certificate status, i.e. the Certification instance will have one of the Life Cycle states derived from the Certification Model; in our scenario the states are ISSUED, SUSPENDED, EXPIRED or REVOKED, there is no NOT_ISSUED state since a Certificate is always created when evidence is valid, in other words there is pending state where the Certificate is issued but without a validity period,
- the ExpirationDate element represents the validity period of the certificate,
- the Assertion that will be better detailed in the next Section,
- the Signature represents the digital signature of the Certification Authority or of another Actor involved in the process,

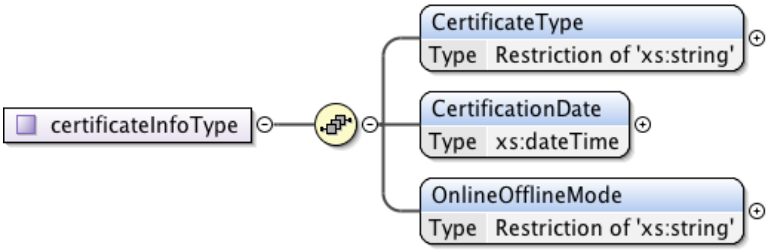


FIGURE 25 – TEST-BASED CERTIFICATE: CERTIFICATE INFO TYPE

An excerpt of the XMD schema is shown below.

```
<xs:element name="TestBasedCertificate" type="testBasedCertificateType"> </xs:element>
```

```
[...]
    <xs:complexType name="testBasedCertificateType">
        <xs:sequence maxOccurs="1">
            <xs:element name="CertificateId" type="xs:integer"/>
            <xs:element name="CertificationModelId" type="xs:integer"/>
            <xs:element name="CertificateInfo" type="certificateInfoType"> </xs:element>
            <xs:element name="CertificateStatus">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="NOT_ISSUED"/>
                        <xs:enumeration value="ISSUED"/>
                        <xs:enumeration value="SUSPENDED"/>
                        <xs:enumeration value="REVOKED"/>
                        <xs:enumeration value="EXPIRED"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ExpirationDate" type="xs:dateTime"/>
            <xs:element name="Assertion" type="assertionType"/>
            <xs:element name="Signature" type="signatureType"/>
        </xs:sequence>
    </xs:complexType>
[...]
```

4.2.1. Assertion

The Assertion element of the XML schema contains the reference to the SecurityProperty (called TestedSecurityProperty), the constraint on the performance attributes of the Security Property detailed by a XPath expression and the Evidence element.

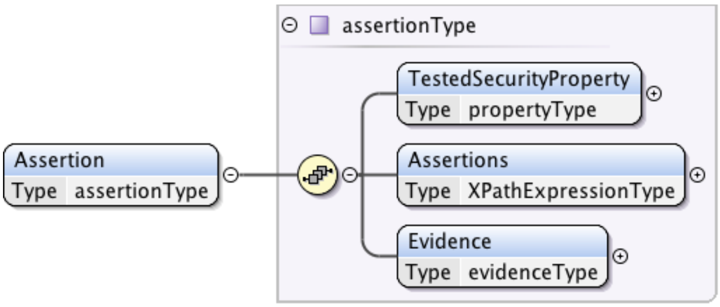


FIGURE 26 – TEST-BASED CERTIFICATE: ASSERTION

Our schema for the Assertion element is shown below.

```
<xs:element name="Assertion" type="assertionType"/>
[...]
    <xs:complexType name="assertionType">
        <xs:sequence>
            <xs:element name="TestedSecurityProperty" type="propertyType"/>
            <xs:element name="Assertions" type="XPathExpressionType"/>
            <xs:element name="Evidence" type="evidenceType"/>
        </xs:sequence>
    </xs:complexType>
```

5. Monitoring-based Certification Model

This section corresponds to section 5 in D2.2. There have been several changes to v1 which are indicated in the individual subsections below.

5.1. Overview

This section corresponds to initial part of Section 5.1 in D2.2. The main amendment is the description of the mapping of monitoring based certification models onto the CUMULUS meta model.

In incremental monitoring based certificates, the evidence required for assessing and verifying security properties is acquired through continuous monitoring of the operation of cloud services. Hence, the evidential basis in such certificates can cover contextual conditions that might not be possible to envisage, test or simulate through other forms of assessment (e.g., testing and static analysis) before deploying a cloud service. Continuous monitoring can capture contextual conditions in cloud service provision as, for example, changes in the population of co-tenant services, the deployed virtualization and optimization strategies and mechanisms, and network and middleware configurations in a cloud, which are difficult to take into account in static forms of assessment. It can also capture and adapt to the migration of SaaS cloud services within cloud federations, providing support for the adaptation of monitoring infrastructures when this happens.

As with all types of certificates, the process of creating and managing incremental monitoring based certificates is driven by certification models. The purpose of such models is to define the security property that is 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 (e.g., when certificates can be issued, should be validated, revoked etc.), the agents which will have responsibility for carrying out different parts of the process (e.g. which will be the agent which will acquire the required monitoring evidence, which agent will sign off the certificate etc.).

To enable the definition of certification models for incremental, monitoring based certificates, we have developed a language for specifying monitoring based certification models (shortly referred to as "MBCM" in the rest of this document). This language has been defined by an XML schema that we present in detail in the sub sections below. More specifically, the language 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 validity checks that should be executed before issuing and/or using a certificate etc. In addition, the monitoring based certification models language enables the specification of elements that pertain to monitoring based certificates but do not necessarily apply to other types of certificates, as for example, responding to conflicts that may arise within the body of evidence that is being acquired in order to assess a security property.

The schema for defining monitoring based certification models complies with the CUMULUS Meta-Model. The following table summarises the correspondences between the different elements of the monitoring based certification model schema and the CUMULUS meta-model.

Monitoring based Certification model (MBCM) elements	**Meta-Model classes**	**Notes on the mapping**
Certification Model	Certification Model	There is a direct correspondence between TOC in MBCM and TOC in the CUMULUS meta model.
TOC	Target of	The *Target of certification* (*ToC*) in the CUMULUS meta

Monitoring based Certification model (MBCM) elements	Meta-Model classes	Notes on the mapping
	Certification	model is the entity that is certified. This can be the service under certification (i.e. SaaS layer), the platform deploying services (PaaS layer), the infrastructure hosting platforms and services (IaaS layer), or any combination of the above (see Sect 2.4). This concept is expressed by TOC in MBCMs. Thus there is a direct correspondence between TOC in MBCM and TOC in the CUMULUS meta model. The meta model leaves the way in which a target of certification may be described open to the discretion of different types of certification models. In the case of MBCMs , TOCs are described by an ID and the interfaces that they provide and expect from external parties as interactions via them may be subjected to monitoring.
Life Cycle Model	Life Cycle	An MBCM defines a life cycle model as required by the CUMULUS meta model. This life cycle model determines the basic states in the generation and management of monitoring based certificates and indicates under what conditions these states may be reached and how they may be altered. A life cycle model is a key concept for MBCMs as it establishes explicitly the relation between different elements in the specification of a MBCM and how they may affect the certificates, which are generated from it. This relation is established in a precise manner enabling the life cycle model to function as the algorithm which is given to the CUMULUS framework in order to be executed and generate certificates.
Assertion + AssessmentScheme (ValidityTests + Conflicts + Anomalies)	Metrics/Conditions	The concept of *Metrics/Conditions* in the CUMULUS meta model corresponds to several elements in MBCMs. In particular, the core metrics and conditions regarding the assessment of the property to be certified are formally expressed through the assertion(s) included in the definition of this property. Further metric and conditions are specified in the *AssessmentScheme* element in MBCMs (e.g., length of monitoring periods, minimum number of monitoring events etc.) and the element *ValidityTests* (e.g., conditions about the configuration and deployment of the monitoring components used to acquire the monitoring evidence).
Monitoring Configuration (Component)	Evidence Aggregator, Evidence Collector	The concept of *Evidence Collector* in the CUMULUS meta model corresponds to different types of components that have responsibility for the collection and analysis of the monitoring evidence required for monitoring based certificates (e.g., event collectors, monitors, aggregators). These are specified as part of

Monitoring based Certification model (MBCM) elements	**Meta-Model classes**	**Notes on the mapping**
		the element *Monitoring Configuration* in MBCM.
Security Property	Security Property	The concept of Security Property in the CUMULUS meta model corresponds to the element Security Property in the MBCM schema. In the MBCM schema, however, the description of security properties in the form of attribute-value pairs is optional since it cannot be used in order to drive the monitoring activity required for the generation of monitoring based certificates. Security properties in MBCS are specified through one or more formal assertions. They may also refer to a categorisation scheme, as the scheme introduced in D2.1
	Attribute (part of Security Property)	
	Commitment	This concept does not exist in MBCMs as it is irrelevant for monitoring based certificates. Commitments are implicitly expressed through security properties in MBCMs.
Assertion	Assertion	The concept of Assertion in the meta model corresponds to the element Assertion in a MBCM. However, in the MBCM, an assertion provides a formal specification of the security property that is to be certified and it determines the evidence that must be monitored in order to assess it.
	Evidence (part of Assertion)	An MBCM does not have an evidence-recording element. It defines, however, what evidence needs to be collected. This evidence is recorded in monitoring based certificates.
	Context	This concept of the meta model has no counterpart in an MBCM.
CASignature	Actor (responsible for Activity)	The concept of *Actor* in the meta model corresponds to the element *CASignature* in an MBCM. *CASignature* represents the entity that signs a certificate generated according to MBCM. This entity (e.g., certification authority) has responsibility for the execution of an MBCM and the certificates generated from it although it might not be the entity that has created this MBCM.

Table 5 – Mapping of Monitoring Based Certification Models onto CUMULUS Meta-Model classes

5.2. Non repudiation of cloud storage services: an example of a service to be certified

This section is new.

To exemplify the use of the amended schema for the specification of monitoring based certification models in the following subsections we use as an example the property of non repudiation (NR). Before presenting the detailed specification of different parts of the certification model for NR in this section we give a general introduction to this property in order to enable the reader understand better the certification model of the property.

Non-repudiation is a property of data storage, requiring that when a data owner (consumer) sends a request to a cloud provider for uploading (downloading) data, the data uploading (downloading) transaction should be conducted in a way such that neither the data owner (consumer) nor the cloud data storage provider could deny having participated in a part or the whole of this transaction.

Several protocols have been proposed to realise non-repudiation (e.g., (Feng, 2011) (Gurgens, 2005) (Ma., 2013) (Markowitch, 1999) (Zhou, 1996) (Zhou, An Efficient Non-Repudiation Protocol, 1997)). 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, i.e., a proof of sending the request, and expects to receive evidence of "Non-Repudiation of Receipt" (NRR) from the cloud provider, acknowledging that the specific request was received. Whilst these protocols have been proven to provide NR under given assumptions their implementation can have bugs or suffer from attacks, such as man-in-the-middle, replay or timeline attacks (Feng, 2011). Therefore, certifying the correct implementation of protocols and the robustness of their implementation to these types of attack is necessary for giving cloud customers the assurances required for NR.

In this section we present an enhanced NR Protocol that can be used in clouds, which is based on (Feng, 2011). This protocol covers an uploading and a downloading session, as well as a recovery phase where a trusted third party (TTP) is involved to resolve any dispute between the other parties. The parties involved are a Data Owner ("A"), a Data User ("B"), the Cloud Provider ("C") and a Trusted Third Party (TTP). TTP is involved in the protocol to resolve any disputation between the data owners or users and the cloud provider. The NR protocol in Figure 27 – Non Repudiation protocol for cloud storage services (based on)has three phases: (1) an upload phase, (2) a download phase, and (3) a resolution of disputate (or recovery), as shown in Figure 27.

Before describing these phases, we provide some basic definitions necessary for understanding them:

- *NRO*: Evidence of Non-Repudiation of Origin, sent by a sender to a receiver. The receiver will hold this evidence as a proof in case the sender denies of having sent a specific message.
- *NRR*: evidence of Non-Repudiation of Receipt, sent by the receiver to the sender. The sender will hold this evidence as a proof in case the receiver denies of having received a specific 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: List of data users B_i who are authorised to download and are capable of decrypting a specific message M.
- Seq_i: Unique sequence number of each message. Each message dispatch must be unique in order to prevent replay attacks.
- $EG_B()$: Group encryption scheme known only to members of B_L .
- $E_X(Y)$: asymmetric encryption of message Y with party X's public key.

- $S_X(Y)$: signature of message Y by party X produced by X's private key.
  The three phases of the protocol are described below.

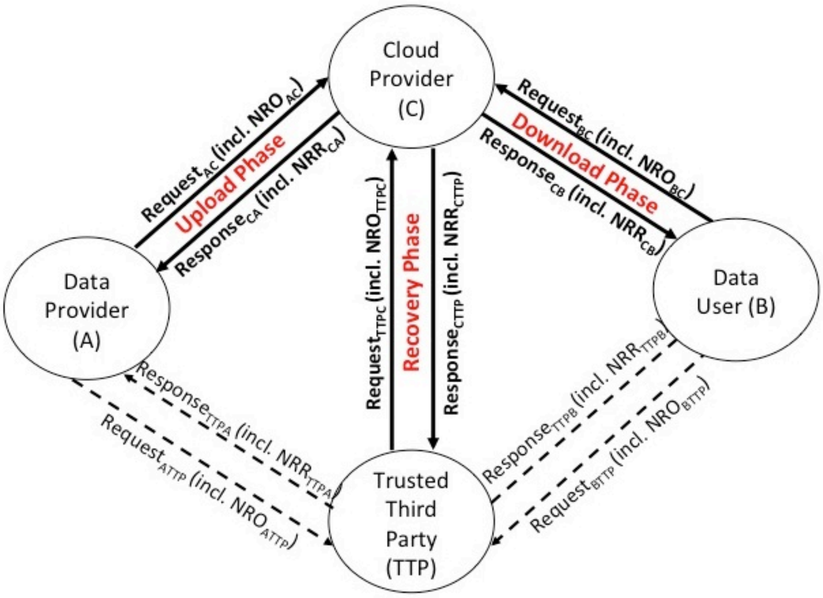

Cloud Provider (C)
Request$_{AC}$ (incl. NRO$_{AC}$)
Upload Phase
Response$_{CA}$ (incl. NRR$_{CA}$)
Request$_{BC}$ (incl. NRO$_{BC}$)
Download Phase
Response$_{CB}$ (incl. NRR$_{CB}$)
Data Provider (A)
Request$_{TTPC}$ (incl. NRO$_{TTPC}$)
Recovery Phase
Response$_{CTTP}$ (incl. NRR$_{CTTP}$)
Data User (B)
Response$_{TTPA}$ (incl. NRR$_{TTPA}$)
Request$_{ATTP}$ (incl. NRO$_{ATTP}$)
Trusted Third Party (TTP)
Response$_{TTPB}$ (incl. NRR$_{TTPB}$)
Request$_{BTTP}$ (incl. NRO$_{BTTP}$)

FIGURE 27 – NON REPUDIATION PROTOCOL FOR CLOUD STORAGE SERVICES (BASED ON (FENG, 2011))

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: NRO_{AB} and NRO_{AC}. NRO_{AB} will be used by data users B to get the key K to decrypt M and $S_A(H(M))$ to verify the data integrity after downloading M from C. A encrypts NRO_{AB} 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 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
  $NRO_{AC}=\{S_A(H(M),H(B_L),EG_B(NRO_{AB}),H(l,Seq_1,T_{g1}, T_1))\}$.

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

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

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
  $NRR_{CA} = \{S_C(H(M)), S_C(H(l, Seq_2, T_{g2}, T_S, NRO_{AC}))\}$.

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 received for M from the data owner A. If B is in B_L, C will send the requested data along with the encrypted NRO_{AB} ($EG_B(NRO_{AB})$) received from A and its own non-repudiation evidence NRR_{CB}. These exchanges are defined below:

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

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))}.

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

Where:

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

  NRR_{CA}={S_C(H(M)),S_C(EG_B(NRO_{AB})),S_C(H(l,Seq_4,T_{g4}))}.

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

Resolution of Dispute

If A does not receive the expected response from the C, it sends a request to TTP with its identification and the NRO_{AC}. TTP will subsequently send this request to C and C should respond with a corresponding NRR_{CA}. The latter exchanges are defined as:

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

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

where:

- T_3 is the maximum time that the sender will wait for an NRR to RQS_{TC}.
- T_{g5} (T_{g6}) is the time of the generation of RQS_{TC} (RSP_{CT}).
- T_S is time when data was stored by C.
- NRO_{TC} is the NRO sent from 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, T_{g5}, NRR_{CA}))}.

5.3. Certification Model XML Schema Description

This section amends section 5.1 in D2.2. The amendment made to this level of the monitoring based certification model are the addition of an element for specifying the target of certification (TOC) and the change of the type of LifeCycleModel elements.

The top layer of v2 of the schema for specifying Monitoring based certification models is shown in Figure 28.

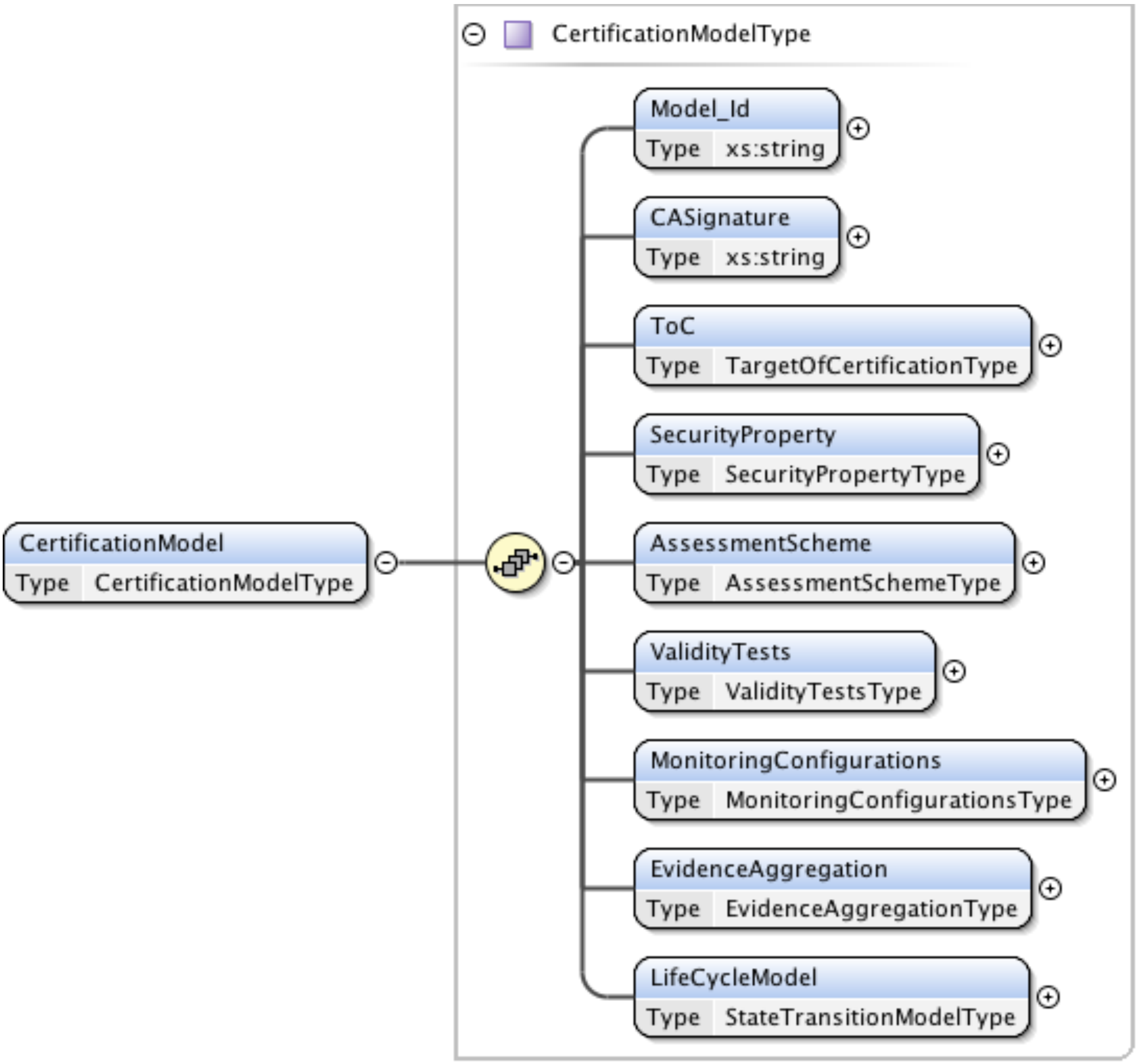


FIGURE 28 – MONITORING-BASED CERTIFICATION MODEL SCHEMA ELEMENTS

As shown in the figure, at the top level there have been two changes with respect to v1 of the schema: (1) a target of certification (TOC) element describing the cloud service(s) to be certified has been introduced at the top level of the model (in v1 of the schema for monitoring based certification models this element was specified as part of the assertions used to define the security property covered by the model) and (2) the type of the element *LifeCycleModel* has been changed to *StateTransitionModelType* (the type of this element in v1 was *LifeCycleModelType*). The meaning and purpose of these elements are defined in the Sections below.

The first of these changes was introduced in order to make monitoring based certification models consistent with the CUMULUS meta-model where the relationship between a certification model and a target of certification is direct (see Figure 1). Furthermore, as a certification model may include security properties defined by more than one assertion, the inclusion of the target of certification element as part of the specification of assertions could lead to redundant and possibly inconsistent specifications of the TOC.

The second change was introduced as we realised that both the expected TOC behaviour model that should be used in order to define evidence sufficiency conditions and the life cycle model that should be used for the specification of monitoring based certificates life cycles could be expressed as state transition machines. Hence, a new monitoring based certification model element, called *StateTransitionModelType* was introduced to cover the specification of both these kinds of models (see Sect. 5.3.6 below for more detail).

5.3.1. Model_Id Element

This section corresponds to section 5.1.1 in D2.2. There are no amendments to this part of the monitoring based certification model.

Model_Id is the element in the schema that 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 target of certification.

5.3.2. CASignature Element

This section corresponds to section 5.1.2 in D2.2. There are no amendments to this part of the monitoring based certification model.

CASignature is the element in the schema that represents the digital signature of the certification authority that has defined or advocated the certification model. *CASignature* is an element of type string. An example of a CASignature element is shown below. The example refers to a certification that will be used to generate certificates which will be signed off by *CUMULUS_City*.

```
<CASignature>CUMULUS_City</CASignature>
```

5.3.3. TargetOfCertification (TOC) Element

This section is new. TOC was defined inside assertions in v1 of the certification model schema. In v2 it is defined as a separate element directly under the certification model.

The *TargetOfCertification* element describes the cloud service to be certified by the particular instance of the certification model. *TargetOfCertification* is an element of type *TargetOfCertificationType.* As shown in the Figure 30 below, the specification of a *TargetOfCertification* includes:

(1) An attribute, called "*id*", which represents the unique identifier of the TOC. This attribute is mandatory.

(2) 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 target of certification offers itself. The *requiresInterface* elements specify interfaces that the target of certification expects an external entity to have.

In the case of the NR property, the target of certification is the cloud provider. The interface that provides this entity is the set of operations of uploading and downloading data, and handling dispute resolution requests by the trusted third party. Assuming the the acknowledgement of results by the cloud provider to data providers, consumers and the trusted third party is asynchronous, the TOC in the case of NR would require three external interfaces: (1) an interface from the data provider to process NRR_{CA}; (2) an interface from the data consumer to process NRR_{CB}; and (3) an interface from the trusted third party to process $NRR_{CT.}$

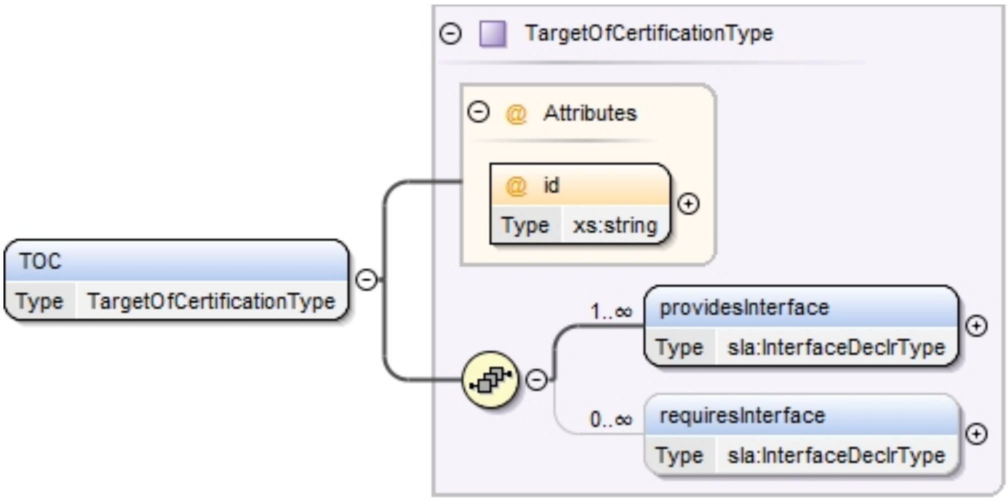

TargetOfCertificationType
Attributes
id
Type xs:string
TOC
Type TargetOfCertificationType
1..∞
providesInterface
Type sla:InterfaceDeclrType
0..∞
requiresInterface
Type sla:InterfaceDeclrType

FIGURE 29 – MONITORING-BASED CM: TARGET OF CERTIFICATION TYPE

The XML schema for specifying *TargetOfCertification* elements in MBCM 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>
```

An example of the *TargetOfCerification* in XML is given below:

```
<TOC id="id">
       <providesInterface>
           [...]
       </providesInterface>
   <requiresInterface>
           […]
       </requiresInterface>
</TOC>
```

The sub-elements of a *TargetOfCertification* specify how interfaces are described the SLA schema and are described below.

- The *providersInterface* element, as shown in Figure 30, includes a sequence of *texts* and *properties* that define the interface that a TOC itself realises. The *providersInterface* defines what operations will be invoked.

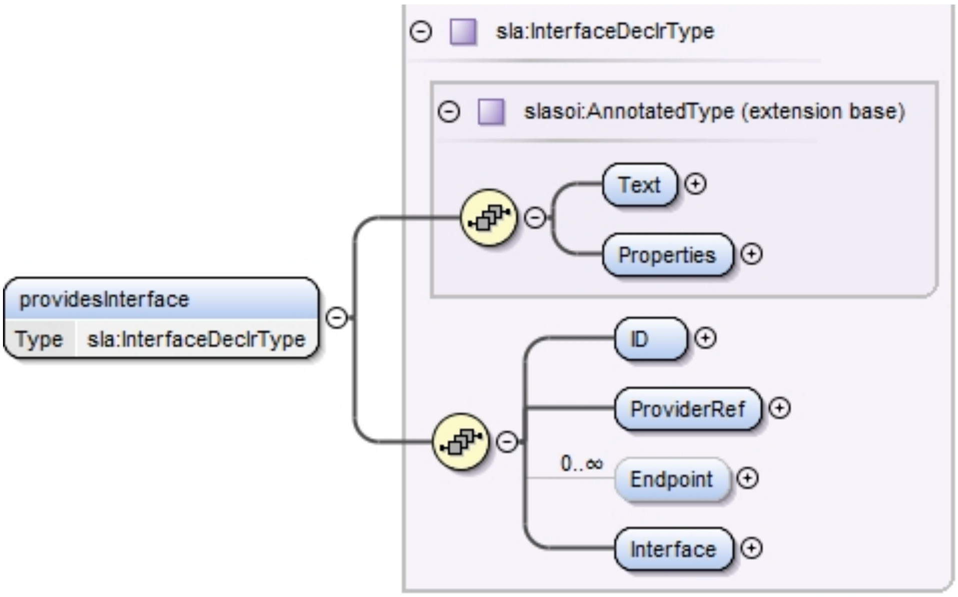


FIGURE 30 – MONITORING-BASED CM: PROVIDESINTERFACE TYPE

- The *requirersInterface* element, as shown in Figure 31, includes a sequence of *IDs*, provider references (*ProviderRef*), zero or more *Endpoints* and *Interfaces*. This element defines the interfaces that the TOC requires from external entities that TOC assumes in order to be able to realise the functionality that will be monitored during the crtification process. To this end, for these interfaces it is important to specify the endpoint where the relevant operations can be invoked.

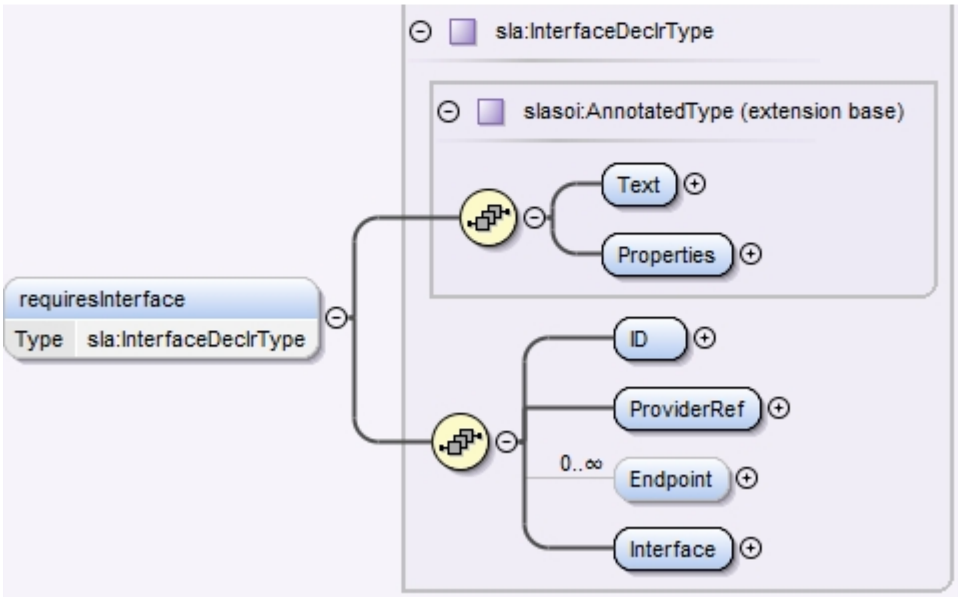


FIGURE 31 – MONITORING-BASED CM: REQUIRESINTERFACE TYPE

Below is an example of *providesInterface* and an example of requiresInterface in XML.

```
     <providesInterface>
       <Text></Text>
       <Properties>
       </Properties>
       <ID>interface::id::c::1/</ID>
       <ProviderRef>nr::id::c</ProviderRef>
       <Endpoint>
         <Text> </Text>
         <Properties> </Properties>
         <ID>c111</ID>
         <Location>http://www.cumulus-project.eu</Location>
         <Protocol> SOAP </Protocol>
       </Endpoint>
       <Interface>
         <InterfaceSpec>
           <Text> </Text>
           <Properties>  </Properties>
           <Name>nr::id::c::cloudinterface</Name>
              <Operation>
             <Text>rqsac</Text>
             <Properties> </Properties>
             <Name>data</Name>
           </Operation>
         </InterfaceSpec>
       </Interface>
     </providesInterface>

  <requiresInterface>
       <Text> </Text>
       <Properties></Properties>
       <ID> interface::id::d::1</ID>
       <ProviderRef></ProviderRef>
       <Endpoint>
         <Text> </Text>
         <Properties></Properties>
         <ID>d111</ID>
         <Location> http://www.cumulus-project.eu </Location>
         <Protocol> SOAP</Protocol>
       </Endpoint>
       <Interface>
         <InterfaceRef>
           <Text> rqsbc</Text>
           <Properties> </Properties>
           <InterfaceLocation> http://www.cumulus-project.eu </InterfaceLocation>
         </InterfaceRef>
       </Interface>
     </requiresInterface>
```

5.3.4. SecurityProperty Element

This section corresponds to section 5.1.3 in D2.2. The amendments made to this part of the model are related to the specification of assertions.

SecurityProperty is the element in the schema that defines the security property that is to be certified by the particular instance of the certification model. *SecurityProperty* is an element of type *SecurityPropertyType*. As shown in the Figure 32 below, this type has:

(1) An attribute, called "*Property*_Category", which defines the category of the security property. Security property names are restricted to the names of properties appearing in the security properties catalogue defined in D2.1.

(2) 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*. The definition of this type is based on the secureSLA* language that was introduced in order to specify security properties in a way similar to normal SLAs. SecureSLA* is based on SLA*, a language that was originally developed in the SLA@SOI project in order to express guarantee terms in service level agreements Several amendments have been made to this language to enable the specification of security properties (e.g., the introduction of multi-valued variables, event timestamps, forced executions of actions). Several amendments have been made to this language to enable the specification of security properties (e.g., the introduction of multi-valued variables, event timestamps, forced executions of actions). A full account of these changes will be given in the forthcoming D3.2 deliverable.

According to this type, an assertion has a UUID, the effective period (*EffectiveFrom*, *EffectiveUntil*), when it was agreed (*AggreedAt*), the parties that are involved (*Party, AbstractParty*), as well as the Interface declaration, the variable declaration and the agreement term of the security property, which consists of a sequence of guaranteed terms (see sub-element *Guaranteed*). A guarantee term defines conditions that must be satisfied. Such conditions refer to the interfaces of cloud services and may make use of one or more variables. Where necessary, further interfaces may also be defined in the assertion element.

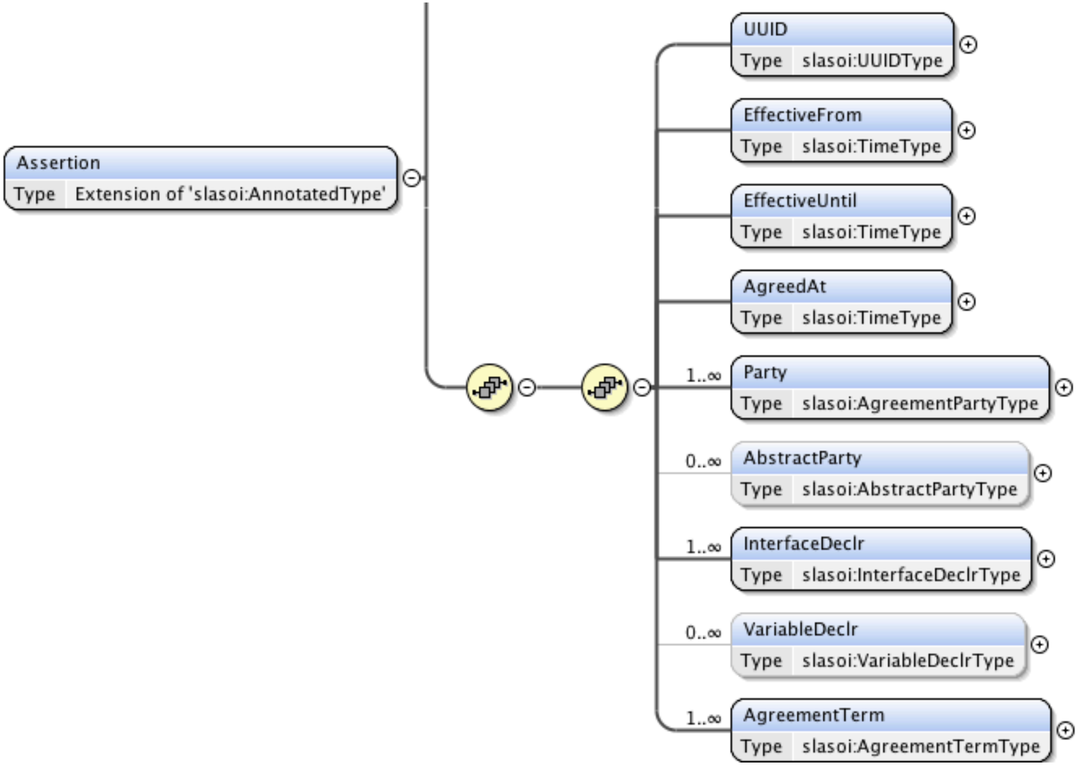


FIGURE 32 – MONITORING-BASED CM: ASSERTION ELEMENT TYPE

The XML Schema of the *SecurityProperty* Element is given in below.

```
<xs:element name="Assertion">
  <xs:complexType>
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="UUID" type="slasoi:UUIDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="EffectiveFrom" type="slasoi:TimeType" />
          <xs:element minOccurs="1" maxOccurs="1" name="EffectiveUntil" type="slasoi:TimeType" />
          <xs:element minOccurs="1" maxOccurs="1" name="AgreedAt" type="slasoi:TimeType" />
```

```
            <xs:element minOccurs="1" maxOccurs="unbounded" name="Party" type="slasoi:AgreementPartyType" />
            <xs:element minOccurs="0" maxOccurs="unbounded" name="AbstractParty"
type="slasoi:AbstractPartyType" />
            <xs:element minOccurs="1" maxOccurs="unbounded" name="InterfaceDeclr"
type="slasoi:InterfaceDeclrType" />
            <xs:element minOccurs="0" maxOccurs="unbounded" name="VariableDeclr"
type="slasoi:VariableDeclrType" />
            <xs:element minOccurs="1" maxOccurs="unbounded" name="AgreementTerm"
type="slasoi:AgreementTermType" />
          </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"></xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
```

The following example shows the specification of a security property. The category of this property is "AIS:non-repudiation:non-repudiation-of-origin" as denoted by the value of the *Property_Category* attribute of security property element. The security property element identifies also the assertion that will provide the formal definition of the property. The identifier of this assertion is "2100".

```
<SecurityProperty   Property_Category="AIS:non-repudiation:non-repudiation-of-origin">
<Assertion>
[...]
</Assertion>
```

An example of the parties involved in the NP protocol is presented below, which are the cloud provider c, the data owner a, the data user b and the trused third party t.

```
<Assertion>
 <Text></Text>
 <Properties>
 </Properties>
 <UUID>url2 </UUID>
 <EffectiveFrom>00:00:00:00</EffectiveFrom>
 <EffectiveUntil>23:59:59:59</EffectiveUntil>
 <AgreedAt></AgreedAt>
<AbstractParty>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>c</ID>
  <Role>cloudprovider</Role>
 </AbstractParty>
 <AbstractParty>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>a</ID>
  <Role>dataowner</Role>
 </AbstractParty>
 <AbstractParty>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>b</ID>
  <Role>datauser</Role>
 </AbstractParty>
 <AbstractParty>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>t</ID>
  <Role>trustedthirdparty</Role>
 </AbstractParty>
[…]
</Assertion>
```

The element *InterfaceDecl* (Interface declarations) shown in Figure 33 provides the means by which details of functional interface specifications are represented. An interface declaration element consists of:

- The sub-element *ID* that uniquely identifies the *InterfaceDeclr*. ID is a mandatory element.
- The sub-element *ProviderRef* that uniquely identifies the party that is obligated to provide the interface. *ProviderRef* is a mandatory element.
- The sub-element *Interface* that defines the interface and which may be one of the following:
  - An inline Interface Specification (*InterfaceSpec*),
  - An interface reference (*InterfaceRef*) to an externally located *InterfaceSpec*, identified by the UUID attribute 'interfaceLocation', or
  - A ResourceType (*InterfaceResourceType*), denoting a resource as a service ("Resource-as-a-Service")

  Interface elements are used in cases where the definition of a security property relates to operations that belong to them. 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 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 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 UUID, which records the address of the endpoint, and
  - A *Protocol* element specifying the communication protocol required for invoking the interface operations at the endpoint. This can take as value a reserved string (STDN) denoting specific communication protocols (e.g., SOAP, REST, SSH).

  Endpoint elements indicate where a service is invoked. For monitoring based certificates this information is useful as it will determine where monitoring events should be captured from and, therefore, possible monitoring configurations for acquiring the required monitoring evidence for a certificate.

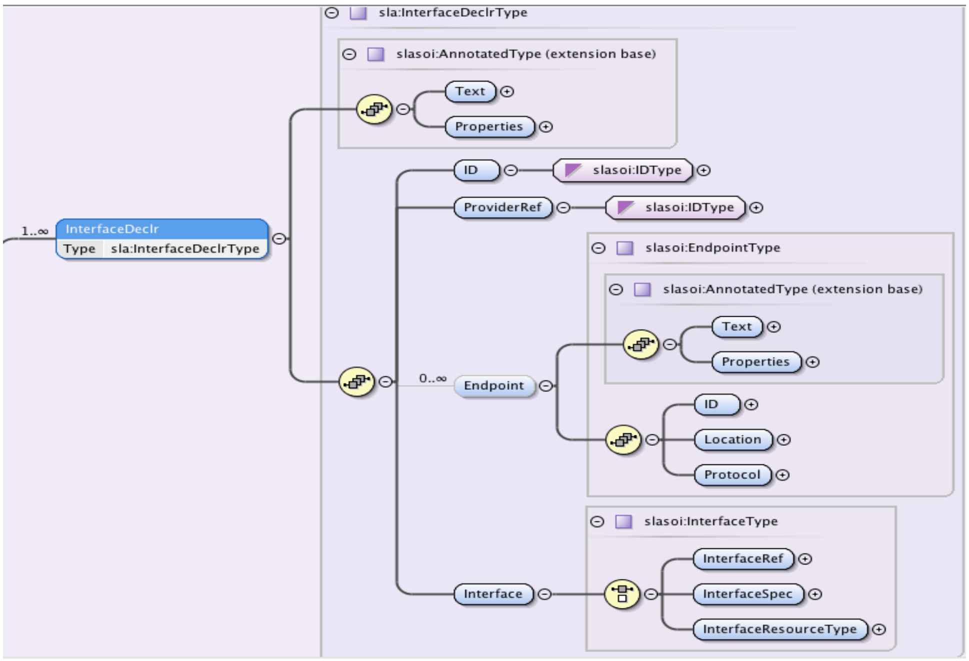


FIGURE 33 – MONITORING-BASED CM: INTERFACE DECLARATION TYPE

An example of an Interface Declaration element for the NR security property is shown below. In this example four interface declarations are defined for each party with the relevant interfaces they require, their provider, as well as the operations of each interface. For instance, the first interface has an ID “c1” and is provided by “c”, as indicated by the ProviderRef sub-element. Moreover, it has an Interface Specification named “cloudInterface” and three operations according to the tree different types of requests that it can receive. The first operation refers to the request made by the data owner “a”, named “rqsac“, which refers to the Upload Phase of the NR protocol. This operation takes as inputs “data” and “t”, which are the data of the request made from the data owner to the cloud provider and the time that it will wait to receive a response. The other two operations refer to the requests made by “b” and “t”, with the names “rqsbc” and “rqstc. These operations refer to the request for downloading data and resolving a dispute and they take as input the data of the request.

```
<InterfaceDeclr>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>c1</ID>
  <ProviderRef>c</ProviderRef>
  <Interface>
   <InterfaceSpec>
    <Text></Text>
    <Properties>
    </Properties>
    <Name>cloudinterface</Name>
    <Operation>
     <Text></Text>
     <Properties></Properties>
     <Name>rqsac</Name>
      <Input>
       <Name>data</Name>
       <Auxiliary>true</Auxiliary>
       <Datatype>url</Datatype>
       <Domain>0</Domain>
      </Input>
      <Input>
       <Text></Text>
       <Properties></Properties>
       <Name>t</Name>
       <Auxiliary>true</Auxiliary>
       <Datatype>url</Datatype>
       <Domain>0</Domain>
      </Input>
    </Operation>
    <Operation>
     <Text></Text>
     <Properties></Properties>
     <Name>rqsbc </Name>
     <Input>
      <Name>data</Name>
      <Auxiliary>true</Auxiliary>
      <Datatype>url</Datatype>
      <Domain>0</Domain>
     </Input>
    </Operation>
    <Operation>
     <Text></Text>
     <Properties></Properties>
     <Name>rqstc </Name>
     <Input>
      <Name>data</Name>
      <Auxiliary>true</Auxiliary>
      <Datatype>url</Datatype>
      <Domain>0</Domain>
     </Input>
    </Operation>
   </InterfaceSpec>
  </Interface>
 </InterfaceDeclr>
 <InterfaceDeclr>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>a1</ID>
  <ProviderRef>a</ProviderRef>
  <Interface>
   <InterfaceSpec>
    <Text></Text>
    <Properties>
    </Properties>
    <Name>ainterface</Name>
  <Operation>
  <Input>
   < Text></ Text>
   < Properties>
   </ Properties>
   < Name>RSPca</ Name>
   < Auxiliary> true </ Auxiliary>
   < Datatype>url</ Datatype>
   <Domain></Domain>
  </Input>
  </Operation>
  </InterfaceSpec>
  </Interface>
 </InterfaceDeclr>
 <InterfaceDeclr>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>b1</ID>
  <ProviderRef>b</ProviderRef>
  <Interface>
   <InterfaceSpec>
    <Text></Text>
    <Properties>
    </Properties>
    <Name>binterface</Name>
    <Operation>
     <Input>
      < Text></ Text>
      < Properties>
      </ Properties>
      < Name>RSPcb</ Name>
      < Auxiliary> true </ Auxiliary>
      < Datatype>url</ Datatype>
      <Domain></Domain>
     </Input>
    </Operation>
   </InterfaceSpec>
  </Interface>
 </InterfaceDeclr>
 <InterfaceDeclr>
  <Text></Text>
  <Properties>
  </Properties>
  <ID>t1</ID>
  <ProviderRef>t</ProviderRef>
  <Interface>
   <InterfaceSpec>
    <Text></Text>
    <Properties>
    </Properties>
    <Name>tinterface</Name>
    <Operation>
     <Input>
     < Text></ Text>
     < Properties>
      </ Properties>
      < Name>RSPct</ Name>
      < Auxiliary> true </ Auxiliary>
      < Datatype>url</ Datatype>
      <Domain></Domain>
     </Input>
    </Operation>
   </InterfaceSpec>
  </Interface>
 </InterfaceDeclr>
```

The example shows an interface with ID “CumulusNRService” that is provided by “CumulusProvider” as indicated by the ProviderRef sub-element of it. Moreover, it has an Interface Specification named “cloudInterface” and an operation named “UploadRequest“, which refers to the Upload Phase of the NR protocol. This operation takes as inputs “RQSac” and “Time_RQSac”, and its output is the “RSPca”, which is the response to the request.

- The VariableDeclr element assigns (or binds) an expression to a specific variable identifier (ID). This identifier can be subsequently used in place of the original expression that is assigned to it Figure 34. Variables are useful as they can be used in defining conditions within assertions, which specify formally security properties. The identifier denoting a variable can, in principle, be bound to any Expr, but of particular use are bindings to the following expressions:
    - Functional expressions,
    - Event expressions, or
    - Service references

A *VariableDeclr* element has also a sub-element, called *Customisable*. This element is used in case there is a need to define some extra constraints of the variable. The element *Expr* that is bound to the variable must in this case be a *DomainExpr*, i.e. an expression specifying a range of permitted values.

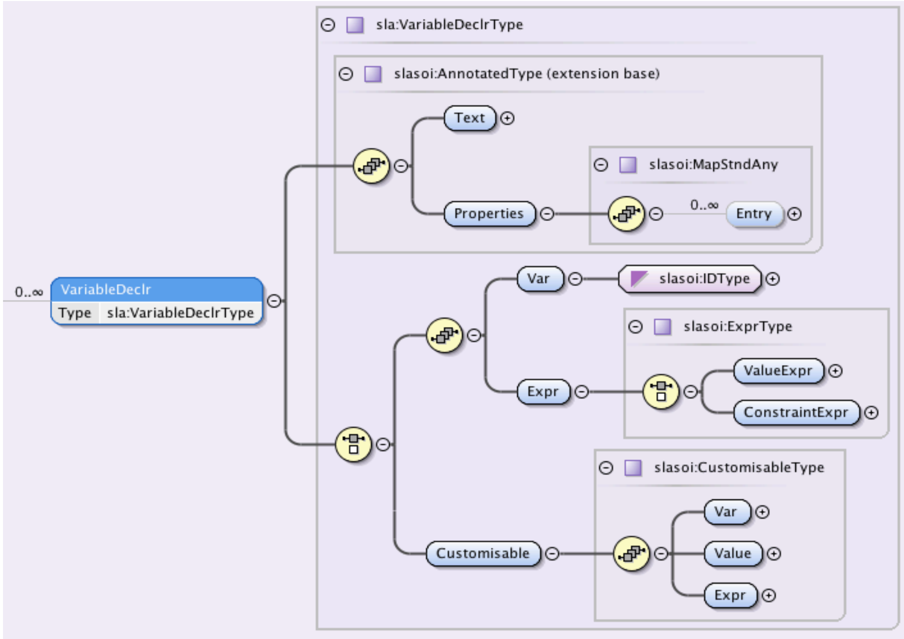


FIGURE 34 – MONITORING-BASED CM: VARIABLE DECLARATION TYPE

An example of a Variable Declaration element is presented below. The example declares the value for the the variable “rqsacv” of the input “rqsac” of the “c1” interface in the interface declaration. In the Expr sub element it is defined that this variable is an Invocation and has a value “nroac”.

```
    <VariableDeclr>
 <Text/>
 <Properties/>
 <Var>rqsacv</Var>
 <Expr>
  <ValueExpr>
   <EventExpr>
    <Text></Text>
    <Properties></Properties>
    <Operator>invocation</Operator>
    <Parameter>
```

```
     <ValueExpr>
      <ServiceRef>
       <OperationList>
        <ID>nroac</ID>
       </OperationList>
       <EndpointList>
        <ID>c111</ID>
       </EndpointList>
      </ServiceRef>
     </ValueExpr>
    </Parameter>
   </EventExpr>
  </ValueExpr>
 </Expr>
</VariableDeclr>
```

- The element *AgreementTerm* within an Assertion element defines a guarantee that a certain state of affairs will hold. This state of affairs is defined in the sub-element *Guaranteed*. Optionally, each *AgreementTerm* element may also be associated with a *Precondition* specifying any conditions that must hold for the state to be guaranteed.

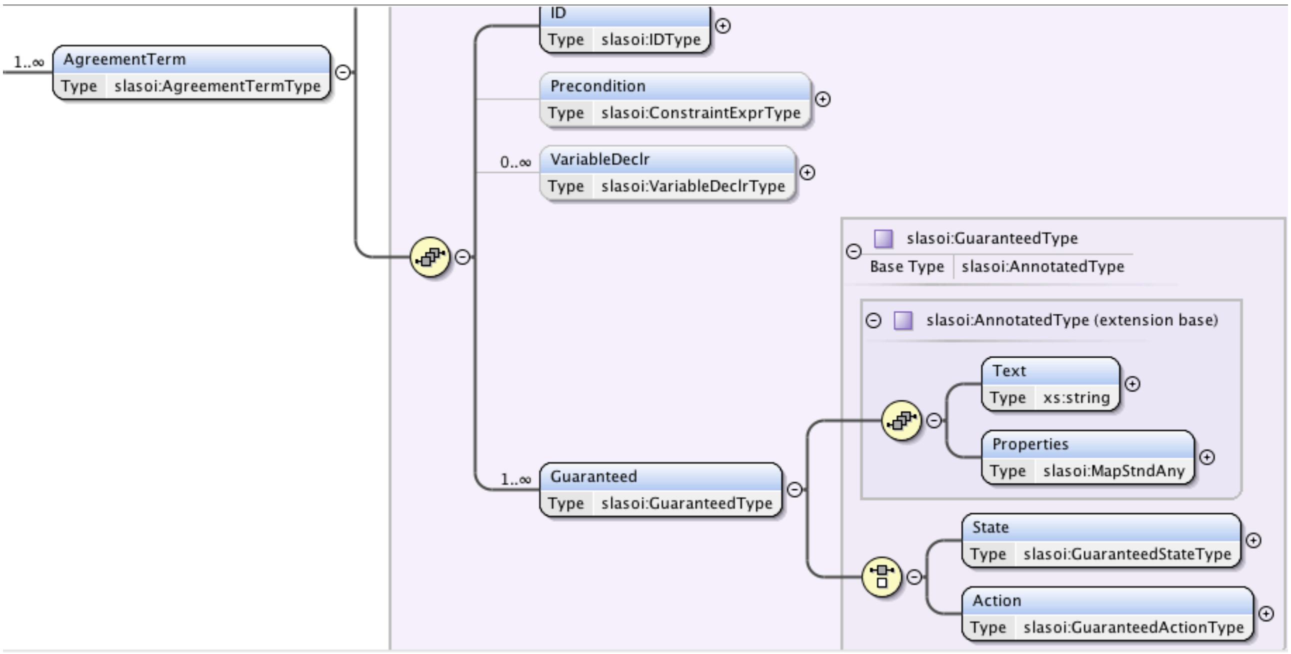

1..∞
AgreementTerm
Type slasoi:AgreementTermType
ID
Type slasoi:IDType
Precondition
Type slasoi:ConstraintExprType
0..∞
VariableDeclr
Type slasoi:VariableDeclrType
slasoi:GuaranteedType
Base Type slasoi:AnnotatedType
slasoi:AnnotatedType (extension base)
Text
Type xs:string
Properties
Type slasoi:MapStndAny
1..∞
Guaranteed
Type slasoi:GuaranteedType
State
Type slasoi:GuaranteedStateType
Action
Type slasoi:GuaranteedActionType

FIGURE 35 – MONITORING-BASED CM: GUARANTEED STATE TYPE

An example of an *AgreementTerm* element is given below.

The *AgreementTerm* in this example has the ID “term1” and a precondition, which states that when a request (“rqsac”) from a data owner (a) for uploading data to a cloud provider (c) is made, there there is no previous request with the same sequence number received. This is expressed by the “Difference” operation, which checks that there is no previous rqsac in rqsac_series matching the latest rqsac invocation (i.e., the value of var RQSacV). Furthermore, the Guarateed element, with the Id “gstate1”, has two “*Constraint*” elements. The first constraint states that the response (“rspca”) should match with the request (“rqsac”). The second constraint states that the time of the response should be less than or equal to the sum of the time of the request and the variable “t”, which is the waiting time of the request to receive the response.

```
<AgreementTerm>
  <Text></Text>
  <Properties></Properties>
  <ID>term1</ID>
  <Precondition>
   <CountExpr>
    <EventExpr>
     <Text></Text>
     <Properties></Properties>
     <Difference>
      <ValueExpr1>
       <ListValueExpr>
        <Value>rqsac</Value>
       </ListValueExpr>
      </ValueExpr1>
      <ValueExpr2>
       <ListValueExpr>
        <Value>rqsacv</Value>
       </ListValueExpr>
      </ValueExpr2>
     </Difference>
    </EventExpr>
    <DomainExpr></DomainExpr>
   </CountExpr>
  </Precondition>
  <Guaranteed>
   <Text></Text>
   <Properties></Properties>
   <State>
    <ID>gstate1</ID>
     <Constraint>
      <FuncExpr>
       <Text></Text>
       <Properties></Properties>
       <Operator>http://www.slaatsoi.org/coremodel#equals</Operator>
       <Parameter>
        <ID>rspcav</ID>
       </Parameter>
       <Parameter>
        <ID>rqsacv</ID>
       </Parameter>
      </FuncExpr>
     </Constraint>
    <Constraint>
     <CountExpr>
      <EventExpr>
       <Text></Text>
       <Properties></Properties>
       <Operator>http://www.slaatsoi.org/coremodel#less_than_or_equal</Operator>
       <Parameter>
        <ConstraintExpr>
         <CountExpr>
          <EventExpr>
           <Text></Text>
           <Properties></Properties>
           <TimeOf>
            <ValueExpr>
             <ID>rspca</ID>
            </ValueExpr>
           </TimeOf>
          </EventExpr>
         </CountExpr>
        </ConstraintExpr>
       </Parameter>
       <Parameter>
        <ConstraintExpr>
         <FuncExpr>
          <Text></Text>
          <Properties></Properties>
          <FuncOp>
           <ArithmeticOp>
            <Add>
             <ValueExpr1>
              <EventExpr>
               <Text></Text>
               <Properties></Properties>
               <TimeOf>
                <ValueExpr>
                 <ID>rqsacv</ID>
                </ValueExpr>
               </TimeOf>
              </EventExpr>
             </ValueExpr1>
             <ValueExpr2>
              <ID>t</ID>
             </ValueExpr2>
            </Add>
           </ArithmeticOp>
          </FuncOp>
         </FuncExpr>
        </ConstraintExpr>
       </Parameter>
      </EventExpr>
     </CountExpr>
    </Constraint>
   </State>
  </Guaranteed>
 </AgreementTerm>
```

5.3.5. AssessmentScheme Element

This section corresponds to section 5.1.4 in D2.2. The assessment scheme specification that it describes has been changed. The changes relate to the specification of evidence sufficiency conditions (conditions regarding the expected behaviour of TOC) and conflicts.

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 sufficiency of evidence collection (e.g., minimum period over which a target of certification must be monitored before a certificate for the particular property of it can be issued), the expiration date of the instance and the absence of conflicting evidence regarding the property to be certified. These conditions must be satisfied, in addition to the guarantee states that are part of the assertion definition of the property, for the certificate to be issued.

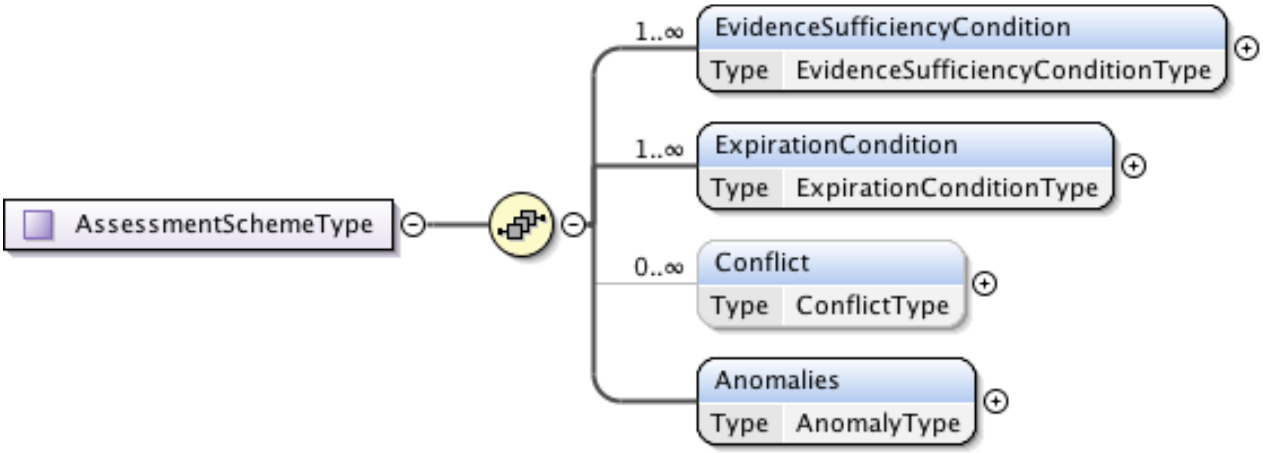


FIGURE 36 – MONITORING-BASED CM: ASSESSMENT SCHEME TYPE

The definition of the type that is used for specifying *AssessmentScheme* elements is shown in Figure 36. As shown in the figure, the specification of an assessment scheme includes a sequence of evidence sufficiency conditions, expiration conditions and conflicts. These sub elements of an assessment scheme are described below.

Evidence Sufficiency Conditions

Evidence sufficiency conditions are conditions regarding the minimum extent and the profile of the monitoring events. This *EvidenceSufficiencyCondition* element 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 listed below:

- *MonitoringPeriodCondition*: A condition of this type can be used to define for how long TOC should be monitored before a certificate can be issued. Such conditions can be specified according to the schema shown Figure 37. It should be noted that the security property to be monitored is determined by the assertions defined as part of the security property 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.

- *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 37. Similarly to the monitoring period element, the monitoring events condition determines the minimum number of events that must have been considered.

- *ExpectedSystemOperationModelCondition:* A condition of this type is used to define an expected operation model of TOC. If such a model is defined the gathered evidence will be deemed sufficient for issuing a certificate only if the actual operation of TOC does not deviate from this model. In v1 of monitoring based certification models this element had not been defined. In v2, 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 along with some margin of error for all these probability measures. Such a model is then used to assess whether the log of events that the monitoring process which assesses a security property for TOC is representative of TOC and therefore sufficient for drawing a conclusion with regards to the satisfaction or not of the property. The exact XML schema for specifying state transition models is presented in Sect. 5.3.6.

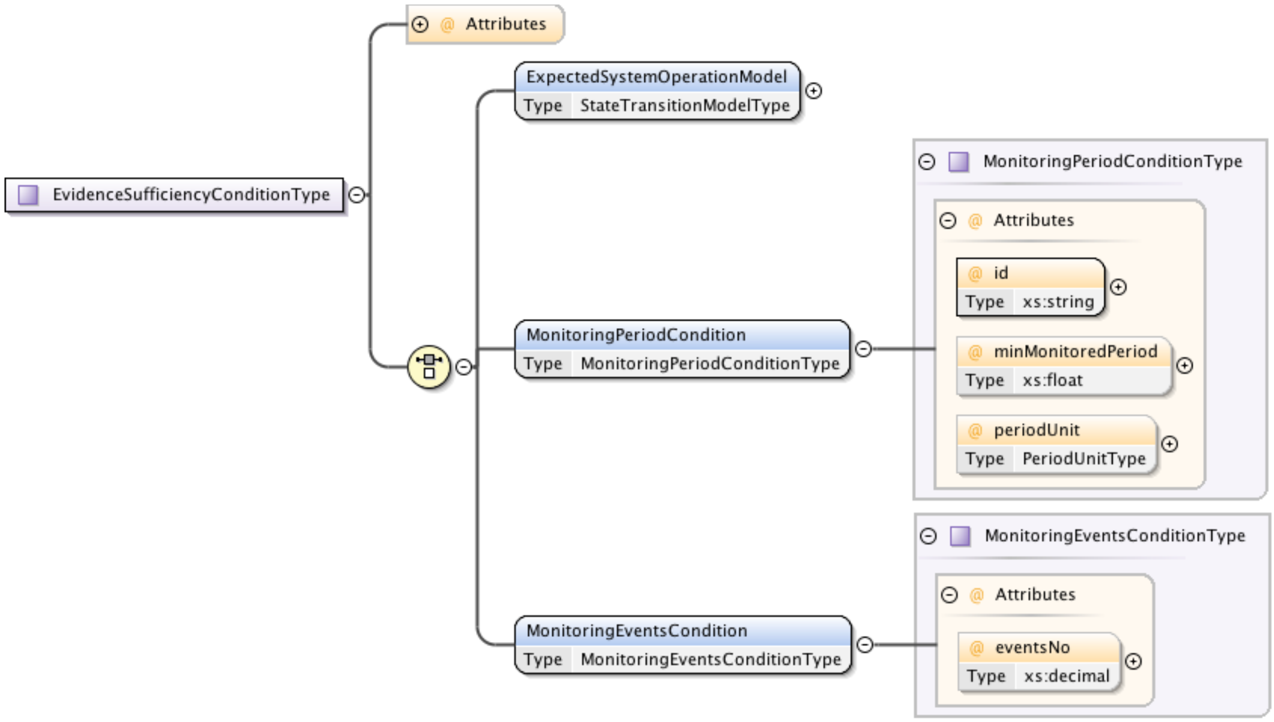

Attributes
ExpectedSystemOperationModel
Type StateTransitionModelType
EvidenceSufficiencyConditionType
MonitoringPeriodConditionType
Attributes
id
Type xs:string
minMonitoredPeriod
Type xs:float
periodUnit
Type PeriodUnitType
MonitoringPeriodCondition
Type MonitoringPeriodConditionType
MonitoringEventsConditionType
Attributes
eventsNo
Type xs:decimal
MonitoringEventsCondition
Type MonitoringEventsConditionType

FIGURE 37 – MONITORING-BASED CM: EVIDENCE SUFFICIENCY CONDITION TYPE

Examples of *EvidenceSufficiencyCondition* elements are given below.

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 720 hours.

```
<EvidenceSufficiencyCondition Id="1011">
        <MonitoringPeriodCondition minMonitoredPeriod="720" periodUnit="hours"/>
</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 it realises the NR protocol and, through it, satisfies NR.

A graphical representation of this model is shown in Figure 38. 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 probability of C receiving such a call whilst being in *state1* is 0.2 with a deviation margin d=0.02 (i.e., a probability between 0.18 and 0.22). This event will make C move to *state2*.

- 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 *state1* is 0.5 with a deviation margin d=0.02 (i.e., a probability between 0.48 and 0.52). This event will make C move to *state4*.
- Calls of the operation *RQStc(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 *state1* is 0.05 with a deviation margin d=0.01 (i.e., a probability between 0.04 and 0.06). This event will make C move to *state3*.
- Calls to an operation *otherO()* that is not related to the NR protocol with a probability Pr=0.25 and a margin d=0.02 (i.e., a probability between 0.248 and 0.252). This event will make C move out of and then back to *state1*.

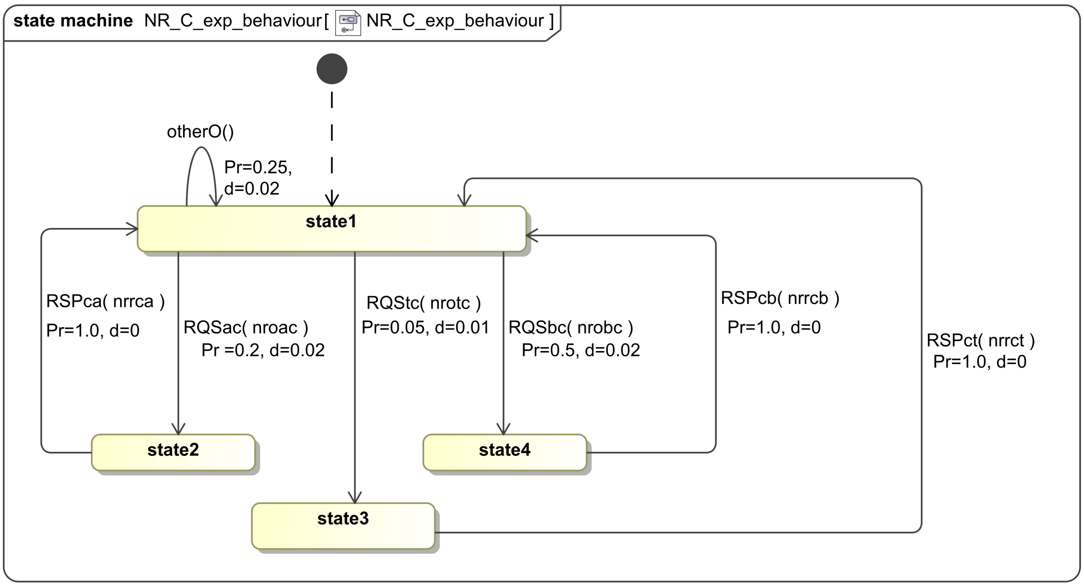


FIGURE 38 – EXAMPLE OF EXPECTED TOC BEHAVIOUR MODEL

The probabilities of the above call events at *state1* reflect the expected use of C, i.e., that (1) it is more likely to receive data download requests than any other request (50%), (2) the next most probable event is the execution of other non NR related operations (25%), (3) data upload requests have a probability that is less than half of data upload requests, and (4) it is relatively unlikely to enter the resolution phase of the NR protocol.

The model also specifies that the probability of C producing an NRR response to A, B or TTP once at states *state2*, *state4* and *state3* respectively is Pr=1.0.

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. Assuming that there have been 10,000 requests to C in the monitoring log, for instance, the expectation set by the model is that there should be: (i) between 2480 and 2520 events irrelevant to the NR protocol (i.e., invocations of the operation *otherO()*), (ii) between 4800 and 5200 data download requests (i.e., invocations of the operation *RQSbc(nrobc)*), (iii) between 1800 and 2200 data upload requests (i.e., invocations of the operation *RQSac(nroac)*), and (iv) between 400 and 600 disputation resolution requests (i.e., invocations of the operation *RQStc(nrotc)*). Hence, if the monitoring log does not contain the above numbers of events, it should not be considered as representative of the use of C and deemed sufficient for issuing a NR protocol certificate.

The specification of different parts of the expected system behaviour model shown in Figure 42 is given in Sect. 5.3.6 after the introduction of the XML schema for the specification of such models.

Expiration Conditions

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 MBCM schema, called *ExpirationCondition*, which is of type *ExpirationConditionType*.

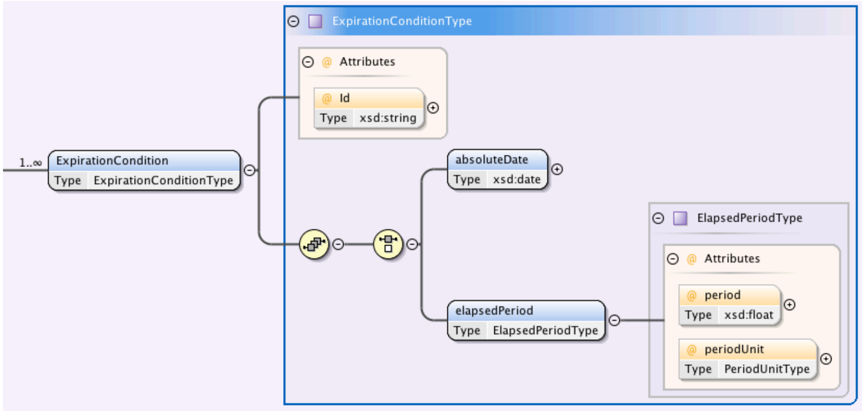


FIGURE 39 – MONITORING-BASED CM: EXPIRATION CONDITION TYPE

The specification of *ExpirationCondition* elements in the MBCM schema is shown graphically in Figure 39. 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 (see attributes period, period unit).

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>
```

Conflicts

In an MBCM 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. The definition of conflict in v2 of MBCMs has been changed with respect to the definition of the "conflict" concept in v1 of such models. More specifically, in v2 of MBCMs, a conflict designates an assessment of the security property associated with the model (MBCM) 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. Consider for instance, an MBCM 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. A conflict in this case would arise if whilst the monthly availability measures of the service satisfy the constraints required by the model (e.g., availability should be at least 95%), if considered over shorter periods (e.g., on a daily or a weekly basis) the measures of the availability of the same service would violate the constraints. For some weeks, the availability of the service may have been found to be less than 95%. Such discrepancies do not necessarily de-validate certificates (e.g., 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.

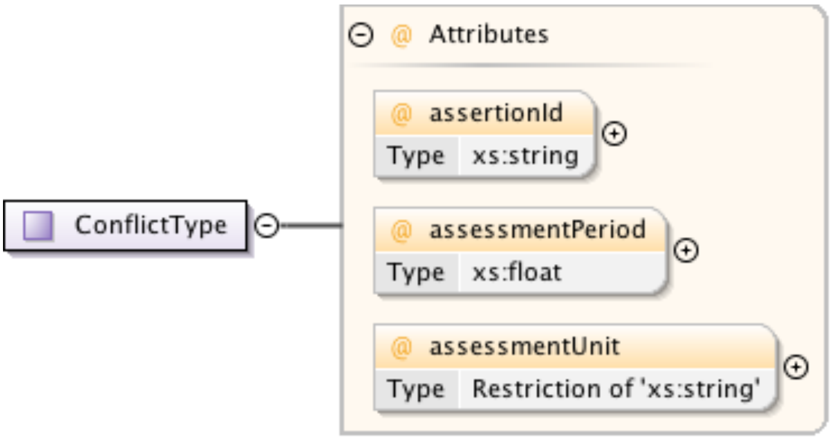


FIGURE 40 – MONITORING-BASED CM: CONFLICT TYPE

A conflict is defined by a *conflict* element in the MBCM schema. These elements are of type *ConflictType*. The definition of *ConflictType* has been changed in v2 of MBCM schema with respect to the definition of the corresponding type in v1 of this schema. The new definition is shown in Figure 40 and its specification in the XML schema is shown below:

```
<xs:complexType name="ConflictType">
  <xs:attribute name="assertionId" type="xs:string"/>
  <xsd:attribute name="assessmentPeriod" type="xs:float"/>
  <xsd: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>
   </xsd:attribute>
  </xs:complexType>
```

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

- *conflictId* – this attribute is a unique identifier of the conflict element
- *assertionId* – this attribute identifies the assertion in the model that the conflict element refers to
- *assessmentPeriod* – this attribute determines the length of the period in time over which the assessment whose purpose is to detect conflicts should be conducted; the value of this attribute is constrained to be less than the normal assessment period of the property in the model by a Schematron rule
- *assessmentUnit* – this attribute determines the time unit in which the conflict assessment period is expressed

An example of a conflict element is given below.

```
<Conflict conflictId="1100" assertionId="guarantee1" assessmentPeriod=1 assessmentUnit=”week”   />
```

The element specifies that a conflict arises when the guarantee element “guarantee1” that is used to assess is violated on the basis of weekly measures. The way to deal with conflicts must also be defined by an MBCM.

The specification of conflict handling is provided as part of the life cycle model of an MBCM (see Sect 5.3.12 below).

5.3.6. StateTransition Model

> *This section relates to Sect. 5.1.8 in D2.2, as it replaces the type LifeCycleModelType in v1 of the schema for specifying monitoring based certification models.*

To specify the expected behaviour of TOC, we use state transition models. These are specified as instances of a new element type that we have introduced to schema for monitoring based certification models, called *StateTransitionModel*. This element type has also replaced the type *LifeCycleModelType* in v1 of the schema that was used to specify life cycle model. The use of *StateTransitionModel* for specifying life cycle models has certain restrictions, which are described in Section 5.3.6 below.

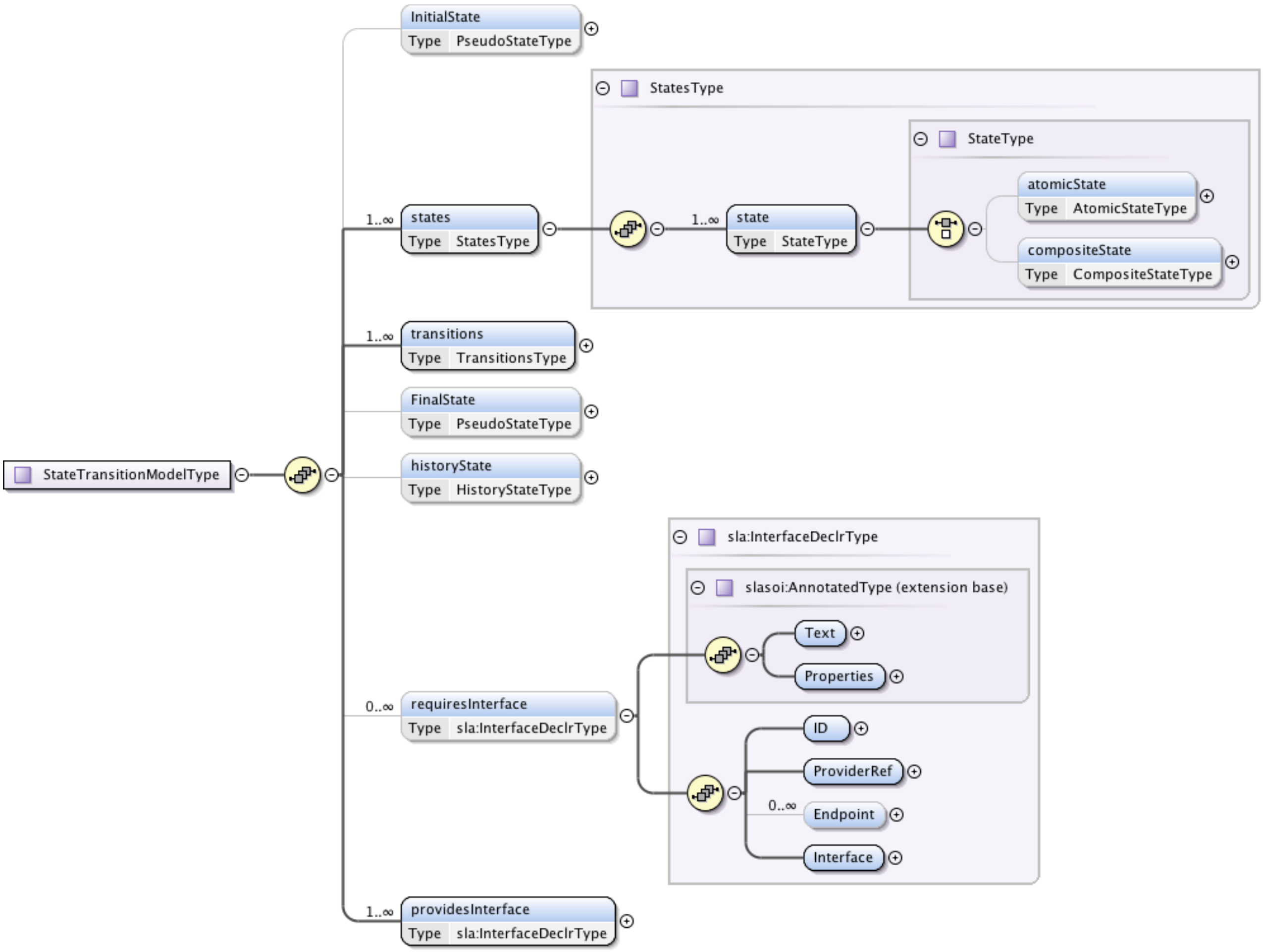


FIGURE 41 – MONITORING-BASED CM: STATE TRANSITION MODEL TYPE

Figure 41 shows a graphical representation of the structure of the XML schema that is used to specify state transition models. As shown in Figure 41 , a state transition model consists of:

- an (optional) *InitialState*
- an (optional) *FinalState*
- an (optional*) HistoryState*
- a sequence of atomic or composite *states,*
- *transitions* between states,
- one or more *provided interfaces*, i.e., sets of operations (aka interfaces) which are realised by the entity whose behaviour is described by the model, and
- one or more *required interfaces*, i.e., sets of operations which the entity whose behaviour is described by the model expects other external interacting entities to have

The specifications of the different elements of a state transition model are described below. However, before presenting the fragments of the MBCM schema for specifying them, we introduce an example.

States

The *InitialState* and *FinalState* elements are pseudo states, which are used to designate (i.e., point to) 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* that is defined as follows:

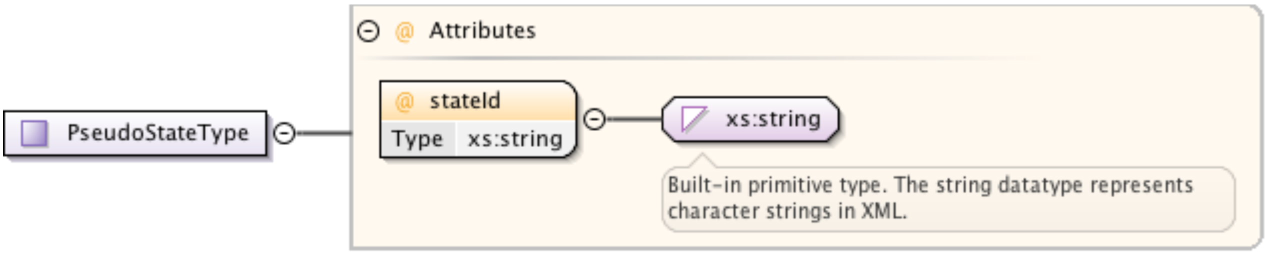

Attributes
PseudoStateType
stateId
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.

FIGURE 42 – MONITORING-BASED CM: PSEUDO STATE TYPE

According to it, pseudo states have only one attribute called *stateId* that uniquely identifies them within a state transition model. The state that the initial pseudo state points to and which point to the final pseudo state are indicated by transitions.

A historic state 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 atomic state within the embedded state transition model that was active before a transition was triggered that moved the state of the relevant entity to state outside from the embedded model that the historic state belongs to and to a state of the embedding model. Historic state elements are described as instances of the element type *HistoricStateType* that is defined below.

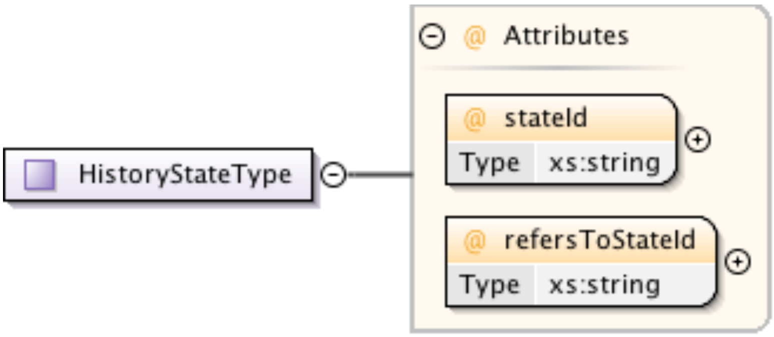


FIGURE 43 – MONITORING-BASED CM: HISTORY STATE TYPE

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 action. Normal states can be atomic or composite.

Atomic states are specified by elements, which are instances of the type AtomicStateType. This type is shown in Figure 44.

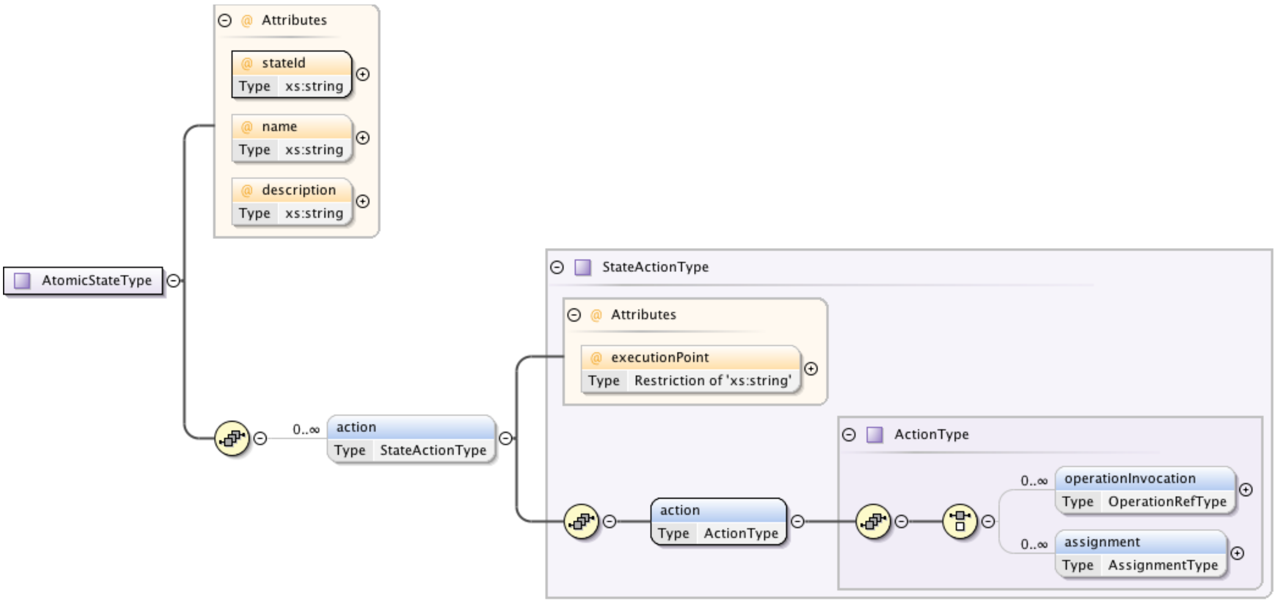


FIGURE 44 – MONITORING-BASED CM: ATOMIC STATE TYPE

As shown in the figure an atomic state element is described by the attributes

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

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 CBCM specification schema, as shown in Figure 44. More specifically, an action is has an attribute, called executionPoint, that 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.

Operation invocation actions are specified according to the type OperationRefType, shown in Figure 45 below.

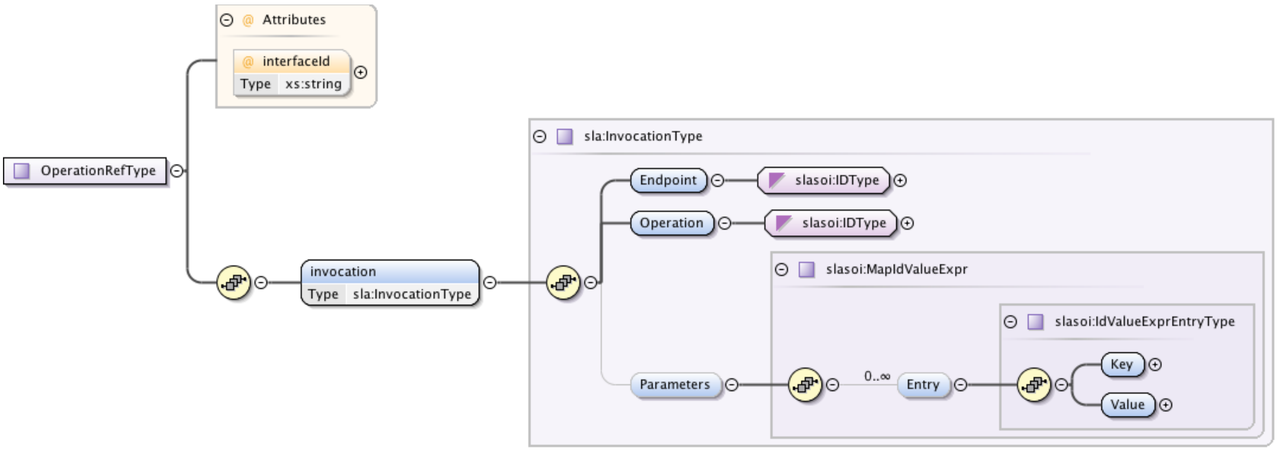


FIGURE 45 – MONITORING-BASED CM: OPERATION REF TYPE

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.
- An *invocation* element. This element is specified by
    - a sub element, called *Endpoint*, which indicates the endpoint where the operation to be invoked 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.

The definition of the type OperationRefType in the MBCM schema is:

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

An example of an atomic state element is shown below:

```
<atomicState stateId="s12657" name="state1" description="state of NR C model">
</atomicState>

<atomicState stateId="s12658" name="state2" description="state of NR C model">
</atomicState>
```

The above examples represent in XML state1 and state2 in the expected behavioural model of the cloud storage service provider C in the NR protocol (see Figure 38).

Composite states are specified as instances of the type *CompositeState*, shown in Figure 46. As shown in the figure, a composite state element has three attributes, namely *stateId*, *name* and *description*. The meaning and use of these attributes is the same as in the case of atomic states. 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 submodel 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 (i.e., OR- or disjunctive-decomposition).

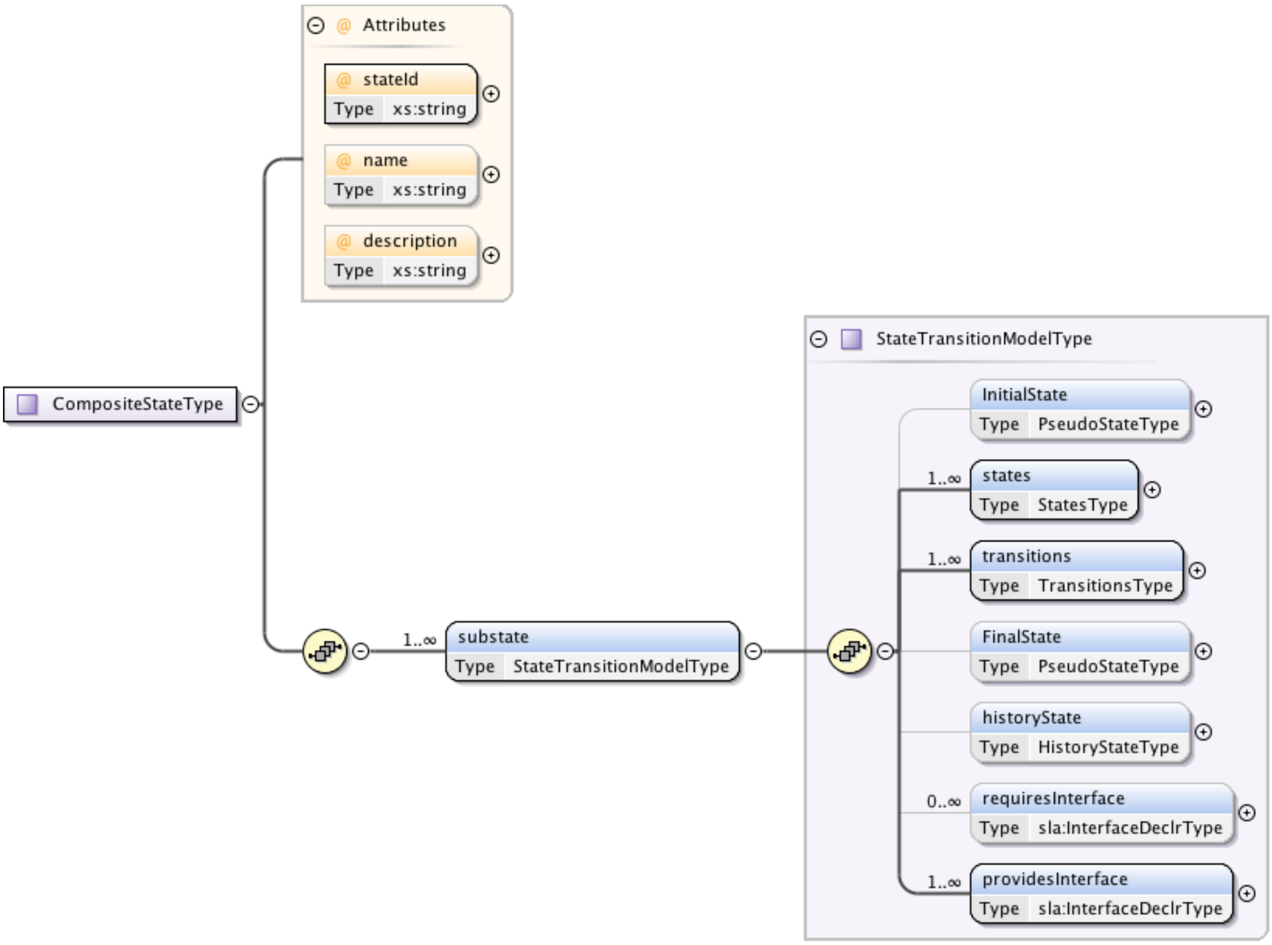


FIGURE 46 – MONITORING-BASED CM: COMPOSITE STATES TYPE

The definition of the type CompositeStateType in the MBCM schema is:

```
<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>
```

Examples of composite states are given in Section 5.3.12 below.

Transitions

As shown in Figure 47, transitions in state transition models are described by a *Transitions* element which of a type *TransitionsType*. This element has one or more sub-elements, which are of type *IndividualTransitionType*.

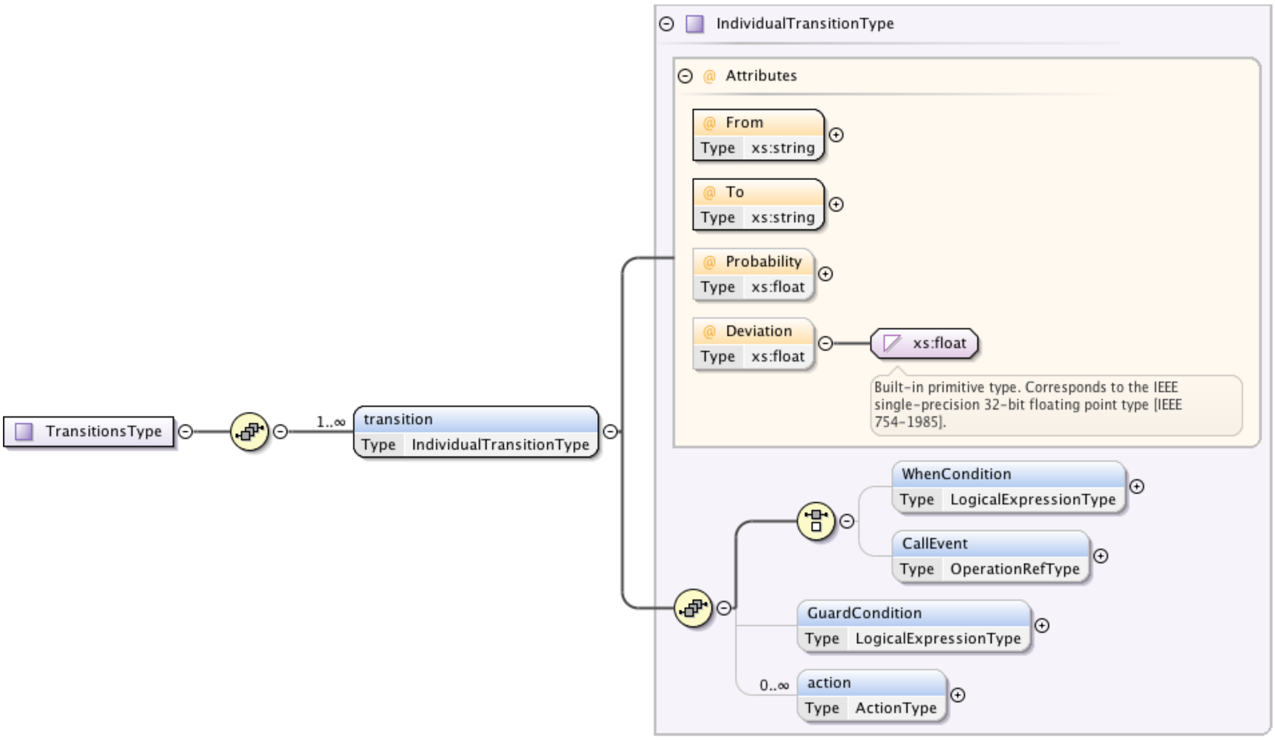


FIGURE 47 – MONITORING-BASED CM: TRANSITION ELEMENT

The definition of type *IndividualTransitionType* 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 CUMULUS 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="Probability" type="xs:float"/>
   <xsd:attribute name="Deviation" 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 starts from
- An attribute of type float, called *Probability*, which indicates the probability that the entity whose behaviour is described by the state transition model will undertake the particular transition if it is in the state indicated by the attribute *From* of the transition. So *Probability* is the conditional probability *Pr(TransitionId|From)*[4]. (NOTE: The attribute *Probability* is used in the case of state transition models which define evidence sufficiency conditions in MBCMs; they are not used in the case of state transition models which define life cycle models in MBCMs).
- An attribute of type float, called Deviation, which indicates the margin of deviation of the probability for the transition.

Transitions can be triggered by two types of events:

- 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 Sect 5.3.7 below. A call event is specified as an element of OperationRefType.

Furthermore, transitions may have:

- A *GuardCondition* element – this is an element expressing a condition which 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.

An example of a transition element is shown below.

```
<transition From="state1" To="state2" Probability="0.2" Deviation="0.02">
  <CallEvent>
    <invocation>
      <sla:Endpoint>b111</sla:Endpoint>
       <sla:Operation><rspcb/sla:Operation>
        <sla:Parameters>
          <sla:Entry>
            <sla:Key>data</sla:Key>
            <sla:Value>nrrcb</sla:Value>
          </sla:Entry>
        </sla:Parameters>
     </invocation>
 </CallEvent>
</transition>
```

The above example represents in XML the transition from state1 to state2 in the expected behavioural model of the cloud storage service provider C in the NR protocol (see Figure 28).

[4] The correct specification of MBCM requires that $\sum_{TrId \in OutTransitions(From)} Pr(TrId|From) \leq 1$.

5.3.7. LogicalExpressionType Element

A logical expression is defined as a condition, or logical combination of conditions, over monitored evidence Figure 48 shows the structure of the LogicalExpressionType and ConditionType.

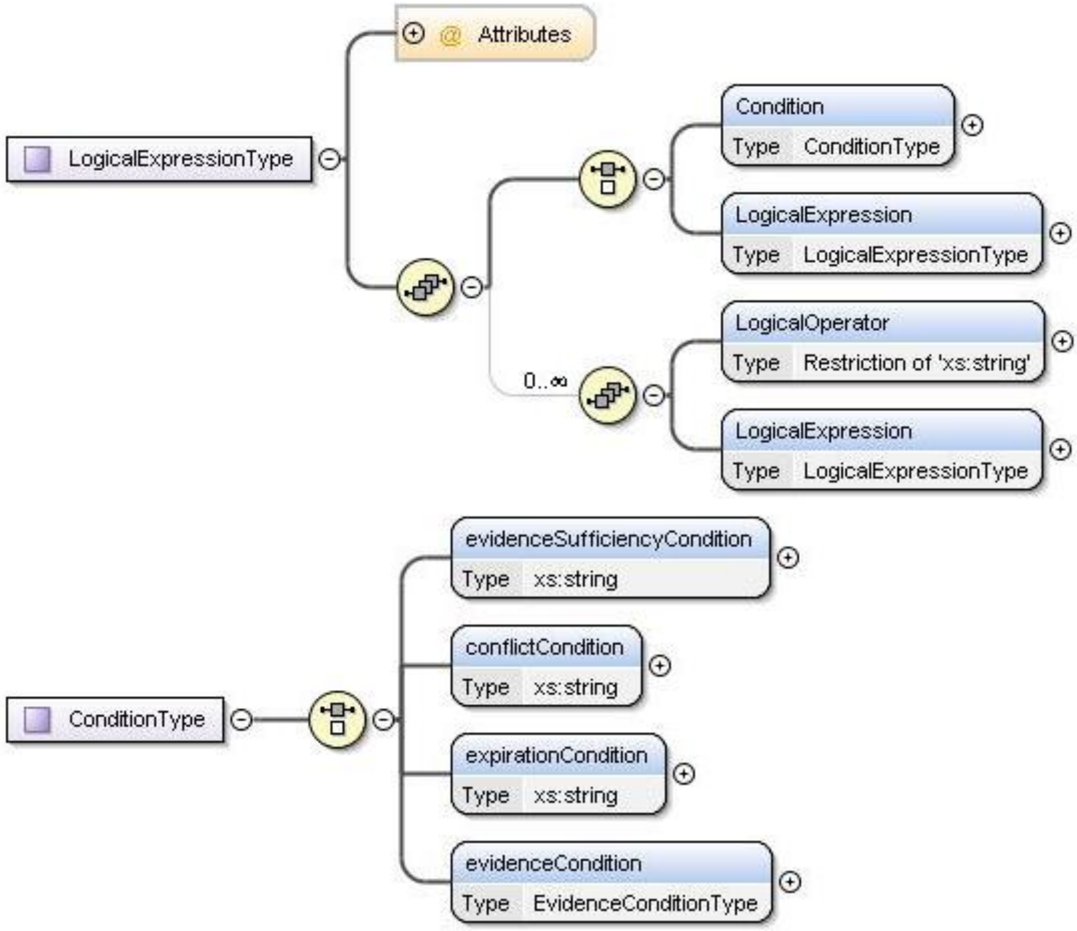


FIGURE 48 – MONITORING-BASED CM: LOGICAL EXPRESSION TYPE AND CONDITION TYPE

The XML schema for the specification of logical expression and condition elements is shown below. As shown in the figure LogicalExpressionType is consists of:

- An attribute that signifies if the logical expression is *negated* or not*,* and
- A choice of a *Condition,* of a type "*ConditionType*", or *LogicalExpression,* of a type *LogicalExpressionType.* Moreover, there is a possibility to have 0 or more LogicalExpression of type *LogicalExpressionType* combined by a *LogicalOperator* (AND/OR). These additional logical expressions can be used in case there is a need to define complex or combined conditions.

XML schema definition shows that *ConditionType* consists of either:

- An *evidenceSufficiencyCondition of type string*, or
- A *conflictCondition of type string,* or
- An *expirationCondition of type string,* or
- An *evidenceCondtion of type EvidenceConditionType*.

It should be noted that *evidenceSufficiencyCondition, conflictCondition* and *expirationCondition* can only be a reference to a predefined condition in the *AssessmentScheme* element; and *evidenceCondition* allows the user to define condition except of the predefined conditions.

```
<xs:complexType name="LogicalExpressionType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="Condition" type="ConditionType"/>
      <xs:element name="LogicalExpression" type="LogicalExpressionType"/>
    </xs:choice>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="LogicalOperator">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="AND"/>
            <xs:enumeration value="OR"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="LogicalExpression" type="LogicalExpressionType"/>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute default="false" name="negated" type="xs:boolean"/>
</xs:complexType>

<xs:complexType name="ConditionType">
  <xs:choice>
    <xs:element name="evidenceSufficiencyCondition" type="xs:string"/>
    <xs:element name="conflictCondition" type="xs:string"/>
    <xs:element name="expirationCondition" type="xs:string"/>
    <xs:element name="evidenceCondition" type="EvidenceConditionType"/>
  </xs:choice>
</xs:complexType>
```

An example condition that refers to previously defined conflict condition is shown below.

```
<Condition>
        <conflictCondition>1100</conflictCondition>
</Condition
```

Figure 49 shows the structure of EvidenceConditionType.

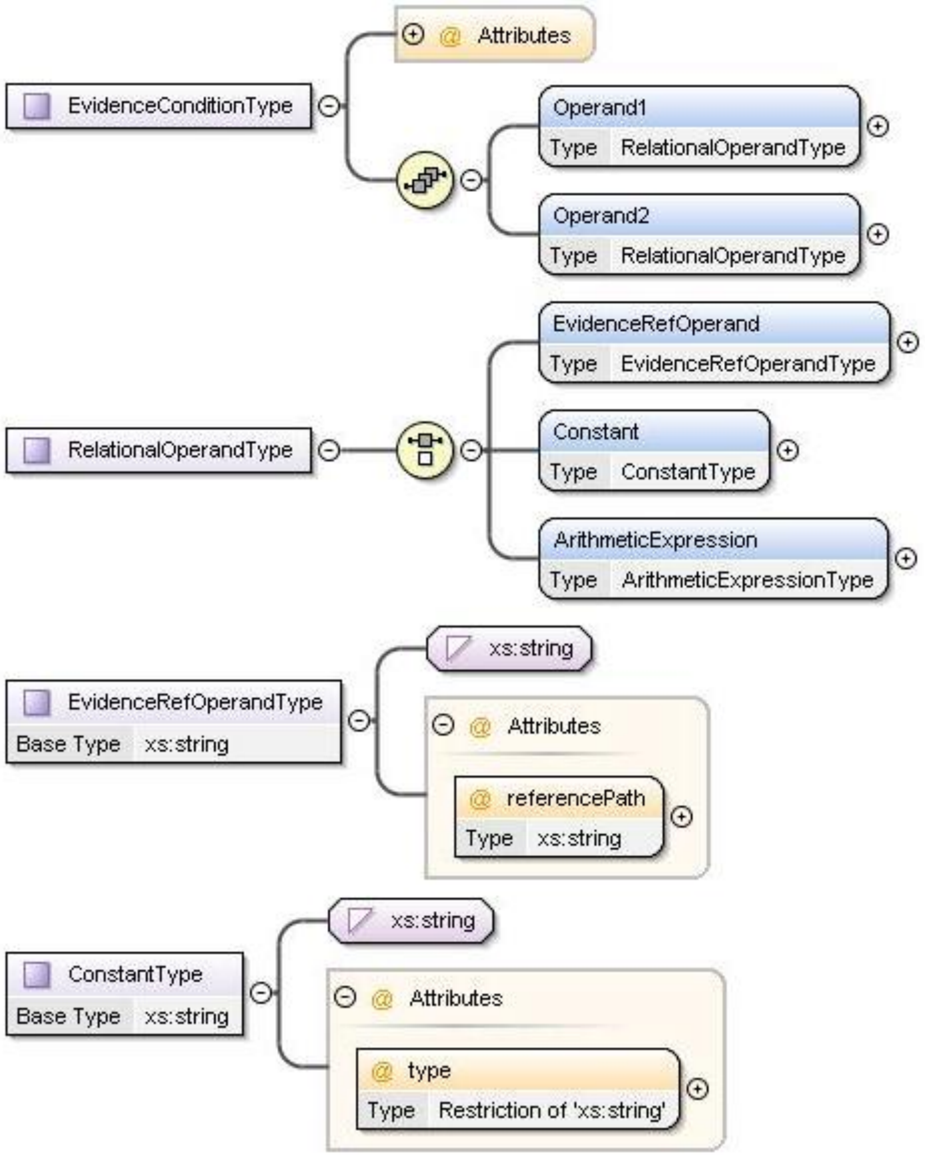


FIGURE 49 – MONITORING-BASED CM: EVIDENCE CONDITION TYPE

The XML schema for the specification of evidence condition elements is shown below. As shown in the figure an evidence condition can be negated (see the attribute *negated*) and is defined as a relational operation (see the attribute *rekation* which can have value *equalTo, notEqualTo, lessThan, greaterThan, lessThanEqualTo, greaterThanEqualTo*) between two operands (see the element *operand1 and* the element *operand2*). These operands can be evidence reference operand, constants, or arithmetic expressions. An evidence reference operand (see the *EvidenceRefOperandType*) can be used to refer to monitorable evidence specified in the certification model. Constant operand can be used to specify a numerical or string constant value. Constant operand can be used define a numerical constant or a string constant (see the *ConstantType* in the figure).

```
<xs:complexType name="EvidenceConditionType">
<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 *evidence condition* is shown below. This condition specifies that the value of the variable *nofcalls* should be equal than 0.

```
<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 monitorabl evidence in certification model. The sturcuture of arithmetic expression is shown in Figure 50 – Monitoring-based CM: Arithmetic Expression Type.

.

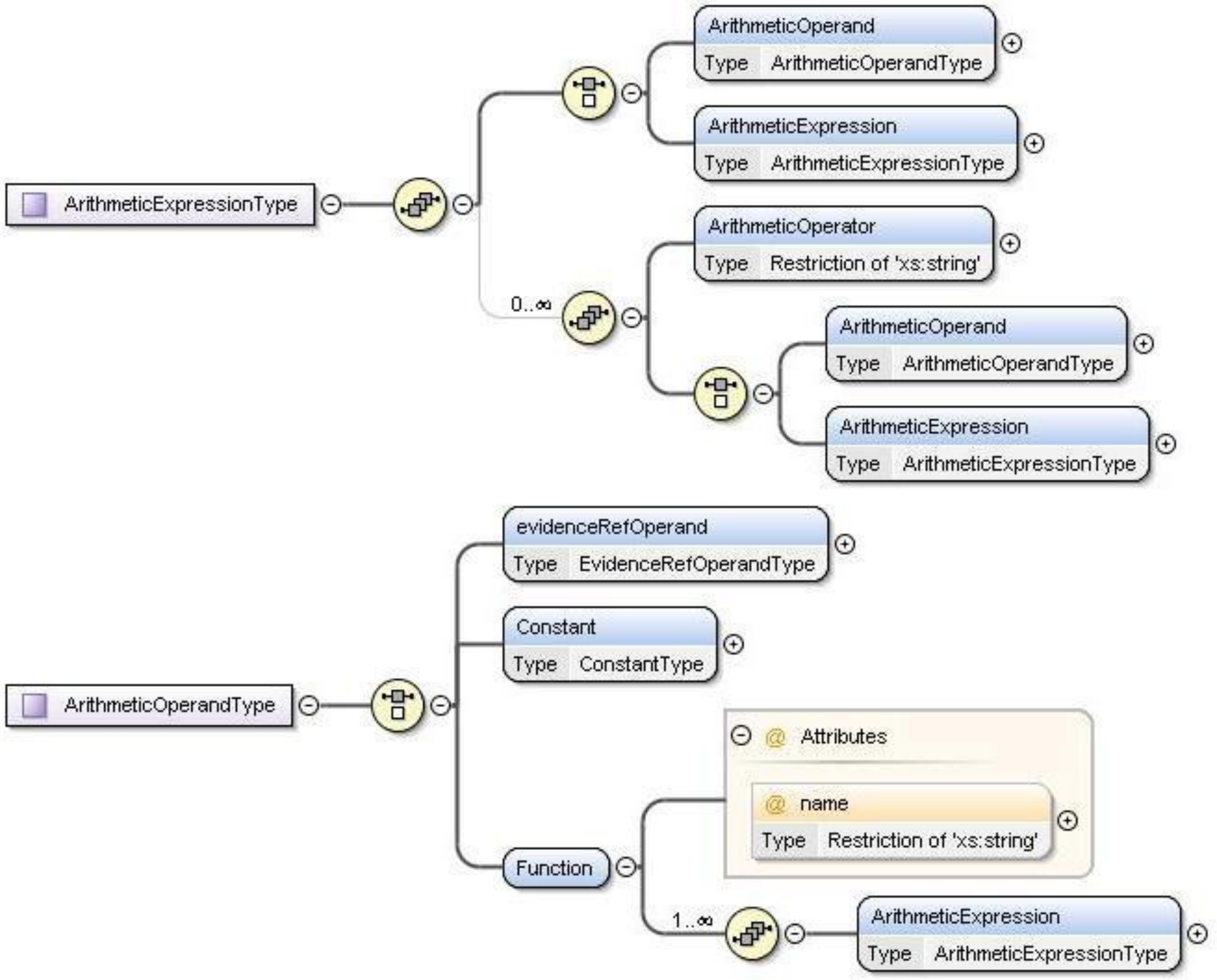


FIGURE 50 – MONITORING-BASED CM: ARITHMETIC EXPRESSION TYPE

The XML schema for the arithmetic expression element is shown below. As shown in the figure arithmetic expression is defined as a sequence of arithmetic operands or other nested arithmetic expressions connected by arithmetic operators. The arithmetic operators are: addition (plus), subtraction (minus), multiplication (multiply), and division (divide) operators. The operands can be evidence reference operands, constants, or functions, as shown in XML schema. 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 those expressions supplied.

```
<xs:complexType name="ArithmeticExpressionType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="ArithmeticOperand" type="ArithmeticOperandType"/>
      <xs:element name="ArithmeticExpression" type="ArithmeticExpressionType"/>
    </xs:choice>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <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:choice>
        <xs:element name="ArithmeticOperand" type="ArithmeticOperandType"/>
        <xs:element name="ArithmeticExpression" type="ArithmeticExpressionType"/>
      </xs:choice>
    </xs:sequence>
  </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 and there no conflict has been detected (see the second condition inside the first nested logical expression) and the value of the variable *nofcalls* is equal to 0 (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>
```

5.3.8. AnomalyMonitoring Element

This section has no corresponding section in D2.2. It describes a new element in the monitoring based certification models that has been introduced in order to specify the monitoring of anomalies.

Certification models may also need to monitor and gather runtime evidence about: (1) potential attacks on TOC, (2) other suspicious behaviour, or (3) operational conditions 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. In the context of monitoring based certification models, we refer to (1)-(3) 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 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.

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 (see Figure 51). The type of this element is *AnomalyType*. The structure of this type is shown in Figure 52 – Monitoring-based CM: Validity Tests Type and the part of the XML schema that defines it is shown below:

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

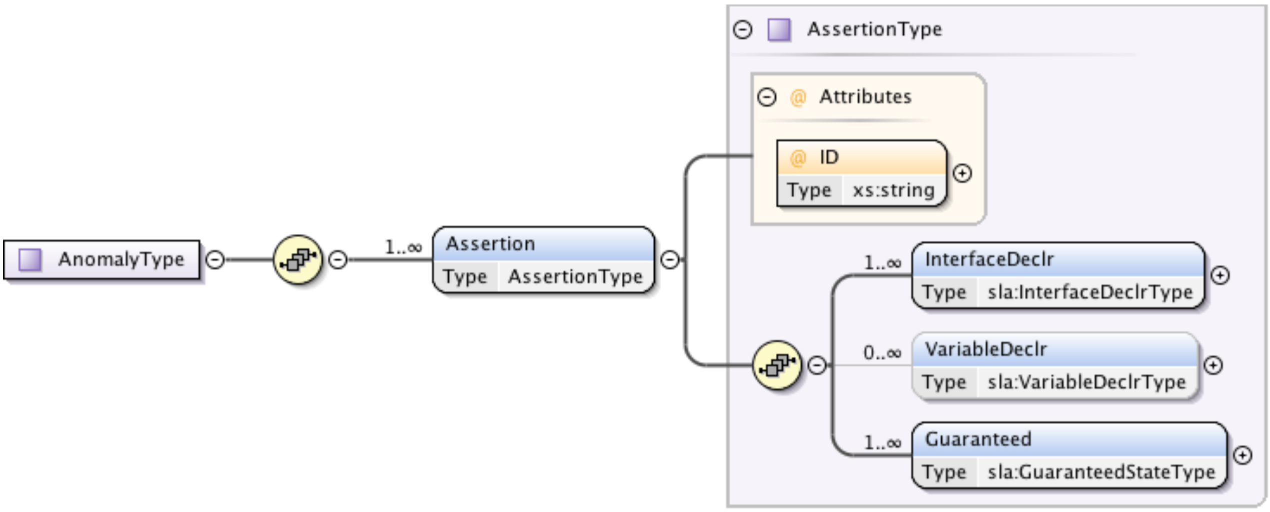


FIGURE 51 – MONITORING-BASED CM: ANOMALY TYPE

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 to be certified whose violation would typically either prevent the issuing of a certificate or lead to the revocation of an issued certificate, anomaly assertion elements are used to gather monitoring evidence indirectly associated with the property that 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 is specified by the life cycle model of the relevant certification model (see Section 5.3.12 for examples of ways of treating anomalies).

In the following we present examples of different anomalies of types (1)-(3) for the certification model of the NR protocol. These exemplify also the use of the anomalies specification schema.

Potential attacks

In the case of NR, As and Bs may be non-trusted parties. Both of them, for instance, may try to launch a denial-of-service attack on 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 R1guaranteed term in the certification model would requires C to respond only to a request from a data provider A 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 this unexpected 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 assertions:

```
<Assertion Id="2101" >
 <Text></Text>
 <Properties>
 </Properties>
 <UUID>url1</UUID>
 <EffectiveFrom>00:00:00:00</EffectiveFrom>
 <EffectiveUntil>23:59:59:59</EffectiveUntil>
 <AgreedAt></AgreedAt>

 <AbstractParty>
  [...]
 </AbstractParty>

 <InterfaceDeclr>
  [...]
 </InterfaceDeclr>

<VariableDeclr>
 [...]
</VariableDeclr>
 <VariableDeclr>
 <Text></Text>
 <Properties></Properties>
 <Var>rsqac</Var>
 <Expr>
  <ConstraintExpr>
   <CountExpr>
    <EventExpr>
     <Text></Text>
     <Properties></Properties>
     <Operator>http://www.slaatsoi.org/coremodel#is_list</Operator>
     <Parameter>
      <ListValueExpr>
       <Value>nroac</Value>
      </ListValueExpr>
     </Parameter>
    </EventExpr>
   </CountExpr>
  </ConstraintExpr>
 </Expr>
</VariableDeclr>

<AgreementTerm>
 <Text></Text>
 <Properties></Properties>
 <ID>term2</ID>
 <Precondition>
  <CountExpr>
   <EventExpr>
    <Text></Text>
    <Properties></Properties>
    <Difference>
     <ValueExpr1>
      <ID>rqsac</ID>
     </ValueExpr1>
     <ValueExpr2>
      <ID>rqsacv</ID>
     </ValueExpr2>
    </Difference>
   </EventExpr>
  </CountExpr>
 </Precondition>
 <Guaranteed>
  <Text></Text>
  <Properties></Properties>
  <State>
   <ID>gstate2</ID>
   <Constraint>
    <FuncExpr>
     <Text></Text>
     <Properties></Properties>
     <FuncOp>
      <ListOp>
    </FuncExpr>
   </Constraint>
  </State>
 </Guaranteed>
</AgreementTerm>
</Assertion>
```

For readability purposes, we also provide the specification of the above anomaly in the BNF syntax of SecurSLA* below[5]:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
         uuid = url2
         sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS --------------------------------------- */

            party {
             id = nr::id::c
             role = nr::cloudprovider
            }

            abstractparty {
             id = nr::id::a
             role = nr::dataowner
            }

            abstractparty {
             id = nr::id::b
             role = nr::datauser
            }

            abstractparty {
             id = nr::id::ttp
             role = nr::trustedthirdparty
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */
```

[5] The specification of the BNF grammar of SecureSLA* is given in Appendix.

```
interfacedecl {
   id = interface::id::c::1
   providerref = nr::id::c

   interfacespec {
       name = nr::id::c::cloudinterface

       operation { name = rqsac
              input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
              input {      name = t
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }


       operation { name = rqsbc
              input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}

       operation { name = rqstc
              input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}

       }}

interfacedecl {
   id = interface::id::a::1
   providerref = nr::id::a

   interfacespec {
       name = nr::id::c::ainterface

       operation { name = rspca
              input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
       }}

interfacedecl {
   id = interface::id::b::1
   providerref = nr::id::b

   interfacespec {
       name = nr::id::c::binterface

       operation { name = rspcb
              input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
       }}

interfacedecl {
   id = interface::id::t::1
   providerref = nr::id::t

   interfacespec {
       name = nr::id::c::binterface
```

```
                    operation { name = rspct
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}
                    }}


/* ---- VARIABLE DECLARATIONS----------------------------------------- */


rqsacv is ( invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ])

rqsbcv is ( invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ])

rspcbv is ( invocation [ invoke { endpoint = b111
       operation = rspcb
       param { name = data value = nrrcb} } ])

rqstcv is ( invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ])

rspctv is ( invocation [ invoke { endpoint = t111
       operation = rspct
       param { name = data value = nrrct} } ])

rqsac::series is series ( nil,
       invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ] )

rqsca::series is series ( nil,
       invocation [ invoke { endpoint = a111
       operation = rqsca
       param { name = data value = nrrca} } ] )

rqsbcv::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ] )

rqstcv::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ] )

rspctv::series is series (
       invocation [ invoke { endpoint = t111
       operation = rspct
       param { name = data value = nrrct} } ] )

rsqac::counts is list[ index = nroac , type(integer) ]

/* ---           Guaranteed Terms     ---------- */

agreement_term { id = at::term::2

       precondition {


       count ( difference [ rqsac::series , rqsacv ] greater_than 0 )
```

```
        }

        guaranteedstate { id = gstate1


        update(rsqac_counts, rsqac_counts::nroac equals rqsacv::data ,
               add ( rsqac::counts, 1))

               }

        }
     }
}
```

The above assertion updates the value of the “rsqac_count” list by adding 1, in case a request rqsac has occurred before. The “rsqac_count” element keeps record of repeated requests made from A to C. Further to the above anomaly-monitoring assumption, the certification model for NR can include a warning to the certification authority regarding the potential compromise of NR as soon as the number of repeated data upload requests exceeds a given threshold (e.g., N repeated data upload requests per minute). This is specified in the life cycle model of the NR certification model (see Sect. 5.3.12 below).

Suspicious behaviour

An example of suspicious behaviour that the NR certification model should monitor is the receipt of requests for recovery from TTP corresponding to requests for data uploading (downloading) from A(B), which have been acknowledged by C. Such requests are suspicious since, in normal circumstances, TTP should not be asking for a recovery of a request that has been acknowledged by C (i.e., a request from A or B for which C has sent an NRR).

This anomalous behaviour from TTP may be due to different reasons. To issue a recovery request, TTP should know the details of the original data uploading (downloading) request from A(B). There are four different ways in which TTP can obtain this knowledge: (i) A or B might have sent the original request to TTP and ask it to initiate the recovery phase, (ii) an attacker, who has managed to obtain the details of the original request of A and B and impersonate them, sends the original request to TTP, or (iii) TTP has itself acted as an attacker (as in (ii)), obtained the details of the original request from A or B and sent the recovery request to C.

Case (i) itself may be the result of a malicious attempt to initiate the recovery phase by A or B. The reason for this could be, for example, to test via TTP how C and TTP would react to such non normal requests and whether it would be possible to launch some DoS attack onto C (via TTP) or onto TTP itself. However, (i) can also be the result of A or B being timed out due to a (non malicious) delay in the arrival of the NRRA(B) sent to them by C caused by the network connection between A(B) and C.

To monitor requests for recovery from TTP corresponding to already responded requests for data uploading from A, the NR certification model uses the following monitoring assumption:

```
<Assertion Id="2102">
 <Text></Text>
 <Properties>
 </Properties>
 <UUID>url1</UUID>
 <EffectiveFrom>00:00:00:00</EffectiveFrom>
 <EffectiveUntil>23:59:59:59</EffectiveUntil>
 <AgreedAt></AgreedAt>
 <AbstractParty>
  [...]
 </AbstractParty>


 <InterfaceDeclr>
 [...]
</InterfaceDeclr>


         <VariableDeclr>
     [...]
    </VariableDeclr>
    <VariableDeclr>
     <Text></Text>
     <Properties></Properties>
     <Var>rpstc_record</Var>
     <Expr>
      <ConstraintExpr>
       <CountExpr>
        <EventExpr>
         <Text></Text>
         <Properties></Properties>
         <Operator>http://www.slaatsoi.org/coremodel#is_list</Operator>
         <Parameter>
          <ListValueExpr>
           <Value>nrotc</Value>
          </ListValueExpr>
         </Parameter>
         <Operator>invocation</Operator>
         <Parameter>
          <ValueExpr>
           <ServiceRef>
            <OperationList>
             <ID>rqstc</ID>
            </OperationList>
            <EndpointList>
             <ID>c333</ID>
            </EndpointList>
           </ServiceRef>
          </ValueExpr>
         </Parameter>
        </EventExpr>
       </CountExpr>
      </ConstraintExpr>
      <ConstraintExpr>
       <CountExpr>
        <EventExpr>
         <Text></Text>
         <Properties></Properties>
         <Operator>http://www.slaatsoi.org/coremodel#equals</Operator>
         <Parameter>
          <ConstraintExpr>
           <CountExpr>
            <EventExpr>
             <Text></Text>
             <Properties></Properties>
             <TimeOf>
              <ValueExpr>
               <ListValueExpr>
                <Value>
                 <ID>rqstc</ID>
                </Value>
               </ListValueExpr>
              </ValueExpr>
             </TimeOf>
            </EventExpr>
       </CountExpr>
      </ConstraintExpr>
     </Parameter>
     <Parameter>
      <ValueExpr>
       <Value>nrotc</Value>
      </ValueExpr>
     </Parameter>
    </EventExpr>
   </CountExpr>
  </ConstraintExpr>
 </Expr>
</VariableDeclr>

<AgreementTerm>
 <Text></Text>
 <Properties></Properties>
 <ID>term3</ID>
 <Precondition>
  <CountExpr>
   <EventExpr>
    <Text></Text>
    <Properties></Properties>
    <Specialisation>
     <ValueExpr>
      <FuncExpr>
       <Text></Text>
       <Properties></Properties>
       <FuncOp>
        <TimeSeriesOp>
         <SeriesValue>
          <FuncExpr>
           <Text></Text>
           <Properties></Properties>
           <Operator>http://www.slaatsoi.org/coremodel#equals</Operator>
           <Parameter>
            <ID>rqstcv</ID>
           </Parameter>
           <Parameter>
            <ID>rqsac</ID>
           </Parameter>
          </FuncExpr>
          <EventExpr>
           <Text></Text>
           <Properties></Properties>
          </EventExpr>
         </SeriesValue>
        </TimeSeriesOp>
       </FuncOp>
      </FuncExpr>
     </ValueExpr>
     <ConstraintExpr>
      <FuncExpr>
       <Text></Text>
       <Properties></Properties>
       <Operator>http://www.slaatsoi.org/coremodel#greater_than</Operator>
       <Parameter>
        <CONST>
         <Value>0</Value>
        </CONST>
        </Parameter>
       </FuncExpr>
      </ConstraintExpr>
     </Specialisation>
    </EventExpr>
   </CountExpr>
  </Precondition>
  <Guaranteed>
   <Text></Text>
   <Properties></Properties>
   <State>
    <ID>gstate3</ID>
    <Constraint>
     <FuncExpr>
      <Text></Text>
      <Properties></Properties>
      <FuncOp>
       <ListOp>
        <Insert>
         <ConstraintExpr>
          <CountExpr>
           <EventExpr>
            <Text></Text>
            <Properties></Properties>
            <Operator>http://www.slaatsoi.org/coremodel#equals</Operator>
            <Parameter>
             <ValueExpr>
              <ID>rsqtc</ID>
             </ValueExpr>
            </Parameter>
            <Parameter>
             <ValueExpr>
              <ID>rqstcv</ID>
             </ValueExpr>
            </Parameter>
           </EventExpr>
          </CountExpr>
         </ConstraintExpr>
         <ValueExpr>
          <EventExpr>
           <Text></Text>
           <Properties></Properties>
           <TimeOf>
            <ValueExpr>
             <ID>rqstcv</ID>
            </ValueExpr>
           </TimeOf>
          </EventExpr>
         </ValueExpr>
        </Insert>
       </ListOp>
      </FuncOp>
     </FuncExpr>
    </Constraint>
   </State>
  </Guaranteed>
 </AgreementTerm>
</Assertion>
```

The specification of the above anomaly in the BNF syntax of SecurSLA* is given below[6]:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
         uuid = url2
         sla_model_version = sla_at_soi_sla_model_v1.0

/* ---- PARTY DESCRIPTIONS (AS ABOVE) --------------------------------------- */

…


/* ---- VARIABLE DECLARATIONS----------------------------------------- */


rqsacv is ( invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ])

rqsbcv is ( invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ])

rspcbv is ( invocation [ invoke { endpoint = b111
       operation = rspcb
       param { name = data value = nrrcb} } ])

rqstcv is ( invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ])

rspctv is ( invocation [ invoke { endpoint = t111
       operation = rspct
       param { name = data value = nrrct} } ])

rqsac::series is series ( nil,
       invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ] )

rqsca::series is series ( nil,
       invocation [ invoke { endpoint = a111
       operation = rqsca
       param { name = data value = nrrca} } ] )

rqsbcv::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ] )

rqstcv::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ] )

rspctv::series is series (
       invocation [ invoke { endpoint = t111
       operation = rspct
```

[6] The specification of the BNF grammar of SecureSLA* is given in Appendix.

```
       param { name = data value = nrrct} } ] )

rqstc_record is list[index = nrotc , invocation [ invoke { endpoint = c333
      operation = rqstc param { name = data value = nrotc} } ] , timeof [invocation [
          invoke { endpoint = c333 operation = rqstc param { name = data value = nrotc} }
] ] ]

/* ---            Guaranteed Terms      ---------- */

agreement_term { id = at::term::3

       precondition {

       /* if there is an nro RQS in RQSac_series matching RQStc */

       count ( specialisation [ rqsac::series ,
                  rqstcv::data equals rqsac::series::data] greater_than 0 )

       }

       guaranteedstate { id = gstate3

       insert(rsqtc::record, rsqtc::record::nrotc equals rqstcv::data ,
              rqstcv, timeof [ rqstcv ])

              }

          }
     }
}
```

The above assertion monitoring formula updates the values of the element “rsqtc_record” list, by inserting the values of rqstcv, if there is a matching request “rqsac” made by A. The “rsqtc_record” is a list variable that keeps record of all requests made by TTP, as well as the time that they were made. Thus, according to the above anomaly-monitoring assumption, the certification model should raise a warning to the certification authority, in case TTP makes a request to C for recovering an acknowledged request from A.

Anomalous operating conditions

An example of an operating condition that should be monitored by the NR certification model is the average time that it takes for a response from C to reach its intended recipient party (i.e., A, B or TTP). Monitoring this time is important as it might indicate that responses to A, B or TTP reach them with delays that can get them timed out, despite C having issued these responses within the time period required by the NR protocol (i.e., within the period [_tAReq,_tAReq+f(_t1)] required by rule R1 in the case of NRRs from C to A). Such delays might be due to network delays or some man-in-the-middle attack on the communication lines between C and A, B and TTP.

Monitoring the exact average time of the arrival of a NRR from C to A, B or TTP is not possible as in general the monitoring framework of the certification authority that realises the NR certification model does not have access to events occurring at A and B. An approximate estimate of this average time is, however, possible by monitoring the average time of network traffic in the opposite direction, i.e., the average time that it takes for RQSAC, RQSBC and RQSTC to reach C after being dispatched by A, B or TTP.

The following anomaly monitoring assertions show how the NR certification model monitors the network delay for traffic from A to C:

```
<Assertion Id="2103">
 <Text></Text>
 <Properties>
 </Properties>
 <UUID>url1</UUID>
 <EffectiveFrom>00:00:00:00</EffectiveFrom>
 <EffectiveUntil>23:59:59:59</EffectiveUntil>
 <AgreedAt></AgreedAt>
 <AbstractParty>
  [...]
 </AbstractParty>
 <InterfaceDeclr>
  [...]
 </InterfaceDeclr>
 <VariableDeclr>
  [...]
 </VariableDeclr>
 <VariableDeclr>
  <Text></Text>
  <Properties></Properties>
  <Var>rqsac_ave</Var>
  <Expr>
   <ValueExpr>
    <ListValueExpr>
     <Value>0</Value>
    </ListValueExpr>
   </ValueExpr>
  </Expr>
 </VariableDeclr>
 <VariableDeclr>
  <Text></Text>
  <Properties></Properties>
  <Var>nofcalls</Var>
  <Expr>
   <ValueExpr>
    <ListValueExpr>
     <Value>0</Value>
    </ListValueExpr>
   </ValueExpr>
  </Expr>
 </VariableDeclr>

 <AgreementTerm>
  <Text></Text>
  <Properties></Properties>
  <ID>term4</ID>
  <Precondition>
   <CountExpr>
    <FuncExpr>
     <Text></Text>
     <Properties></Properties>
     <FuncOp>
      <ArithmeticOp>
       <Divide>
        <ValueExpr1>
         <FuncExpr>
          <Text></Text>
          <Properties></Properties>
          <FuncOp>
           <ArithmeticOp>
    <Add>
     <ValueExpr1>
      <FuncExpr>
       <Text></Text>
       <Properties></Properties>
       <FuncOp>
        <ArithmeticOp>
         <Multiply>
          <ValueExpr1>
           <ID>rqsac_ave</ID>
          </ValueExpr1>
          <ValueExpr2>
           <ID>nofcalls</ID>
          </ValueExpr2>
         </Multiply>
        </ArithmeticOp>
       </FuncOp>
      </FuncExpr>
     </ValueExpr1>
     <ValueExpr2>
      <FuncExpr>
       <Text></Text>
       <Properties></Properties>
       <FuncOp>
        <ArithmeticOp>
         <Substract>
          <ValueExpr1>
           <ID>rqstcv</ID>
          </ValueExpr1>
          <ValueExpr2>
           <CONST>
            <Value>1</Value>
           </CONST>
          </ValueExpr2>
         </Substract>
        </ArithmeticOp>
       </FuncOp>
      </FuncExpr>
     </ValueExpr2>
    </Add>
   </ArithmeticOp>
  </FuncOp>
 </FuncExpr>
</ValueExpr1>
<ValueExpr2>
 <FuncExpr>
  <Text></Text>
  <Properties></Properties>
  <FuncOp>
   <ArithmeticOp>
    <Add>
     <ValueExpr1>
      <ID>nofcalls</ID>
     </ValueExpr1>
     <ValueExpr2>
      <CONST>
       <Value>1</Value>
      </CONST>
     </ValueExpr2>
    </Add>
```

```
           </ArithmeticOp>
          </FuncOp>
         </FuncExpr>
        </ValueExpr2>
       </Divide>
      </ArithmeticOp>
     </FuncOp>
    </FuncExpr>
   </CountExpr>
  </Precondition>
  <Guaranteed>
   <Text></Text>
   <Properties></Properties>
   <State>
    <ID>gstate4</ID>
    <Constraint>
     <FuncExpr>
      <Text></Text>
      <Properties></Properties>
      <Operator>http://www.slaatsoi.org/coremodel#less_than</Operator>
      <Parameter>
       <ID>rqsac_ave</ID>
      </Parameter>
      <Parameter>
       <CONST>
        <Value>30</Value>
       </CONST>
      </Parameter>
     </FuncExpr>
    </Constraint>
   </State>
  </Guaranteed>
 </AgreementTerm>
</Assertion>
```

The representation in SecureSLA* is given below:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
         uuid = url2
         sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS (AS ABOVE) --------------------------------------- */

…

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


rqsacv is ( invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ])

rqsbcv is ( invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ])
rspcbv is ( invocation [ invoke { endpoint = b111
       operation = rspcb
       param { name = data value = nrrcb} } ])

rqstcv is ( invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ])

rspctv is ( invocation [ invoke { endpoint = t111
       operation = rspct
       param { name = data value = nrrct} } ])

rqsac::series is series (
       invocation [ invoke { endpoint = c111
       operation = rqsac
       param { name = data value = nroac} } ] )

rqsca::series is series (
       invocation [ invoke { endpoint = a111
       operation = rqsca
       param { name = data value = nrrca} } ] )

rqsbcv::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsbc
       param { name = data value = nroac} } ] )

rqstcv::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqstc
       param { name = data value = nrotc} } ] )

rspctv::series is series (
       invocation [ invoke { endpoint = t111
       operation = rspct
       param { name = data value = nrrct} } ] )

rqsac_ave is type(float) initially 0 units
nofcalls is type(integer) initially 0 units

/* ---            Guaranteed Terms     ---------- */

agreement_term { id = at::term::3

       precondition {


              rqsac_ave is divide ( add ( multiply ( rqsac_ave, nofcalls ),
              subtract ( rqstcv::data::t::areg , rqstcv::data::t::g::one ) ),
              add ( nofcalls, 1 ))

       }

       guaranteedstate { id = gstate3

              rqsac::ave less_than 30

           }

       }
    }
}
```

This assertion monitoring formula updates the variable rqsav_ave that is used to keep a record of the average time that it takes for data upload requests RQSAC to reach C from A. The variable noofcalls in the assertion is the variable keeping the number of requests that have been taken into account for calculating this average. In this case, the certification model should raise a warning to the certification authority in cases where rqsav_ave > f(t1), as this would lead to A being systematically timed out due to delays in the network traffic between A and C. This warning is also shown in the NR certificate life cycle model in Sect. 5.3.12.

5.3.9. ValidityTests Element

This section corresponds to section 5.1.5 in D2.2. There are no amendments to this part of the monitoring based certification model.

A certification model may, in addition to the assessment scheme, define extra validity tests as preconditions for issuing a certificate of a given type. These tests may relate, for example, to conditions regarding the cloud where the service is deployed (e.g., requiring that the cloud offers full isolation of virtual machines) or the adherence of other services that this service may depend on to standards (e.g., requiring that a storage service, which is used by a SaaS service implements correctly a proof-of-retrievability protocol) or 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 (see Figure 52).

The type for specifying validity tests has not been defined yet. It will be defined in the third year of the project. However, we envision that the specification of validity tests can be based on expressing conditions regarding Trusted Computing based certificates and/or other types of certificates regarding TOC and/or its operational context that will 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 monitoring period. Thus, we expect that validity tests can be expressed as logical conditions over such certificates and their contents.

ValidityTests
Type
ValidityTestsType
ValidityTestsType

FIGURE 52 – MONITORING-BASED CM: VALIDITY TESTS TYPE

5.3.10. MonitoringConfigurations Element

This section corresponds to section 5.1.6 in D2.2. There are no changes to the monitoring based certification model schema but we have provided examples of monitoring configuration elements.

This element specifies the list of the monitoring configurations that have been used to collect the evidence for generating certificates Figure 53).

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: (1) *sensors,* which are components capable of capturing and transmitting primitive monitoring events, and (2) *reasoners,* which are components capable of analysing events and checking whether monitoring conditions are satisfied (aka *monitors*).
- *ConcreteProperty* – This element provides the concrete operational specification of the security property that is to be certified by the model, expressed in the language accepted by the reasoner(s) of the particular monitoring configuration. The concrete security property is generated automatically from the abstract security property once a monitoring configuration is selected.

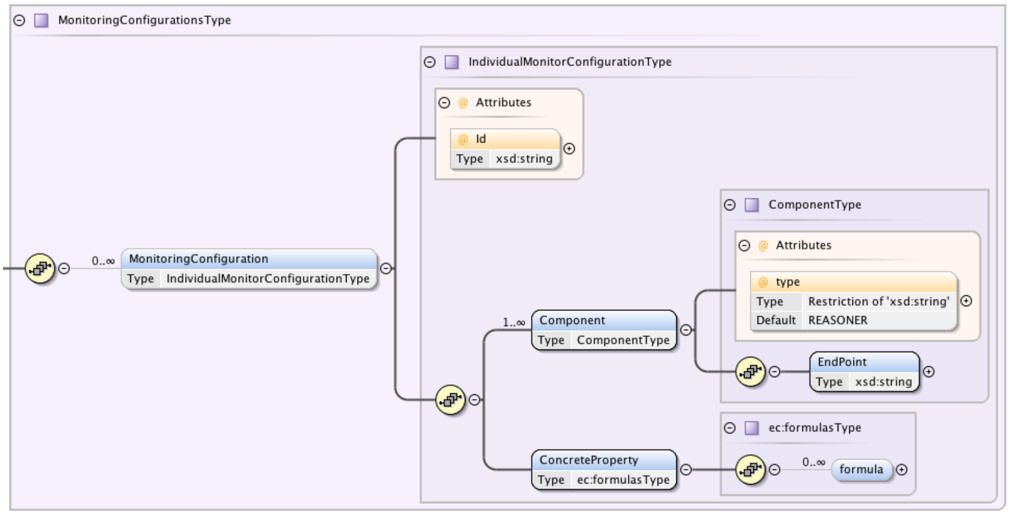


FIGURE 53 – MONITORING-BASED CM: INDIVIDUAL MONITORING CONFIGURATION TYPE

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>
```

An example monitoring configuration is shown below. As show in this example, the framework is configured with reasoner (see the Component element insidet he MonitoringConfiguration element) component and the selected reasoned is assigned with a set of concrete properties (see the formula elements inside the ConcreteProperty element) that should be monitored. For brevity the monitorable formulas are not shown here.

```
<MonitoringConfiguration Id="Id4">
   <Component type="REASONER">
            <EndPoint>https://192.168.43.23:8888/CumulusService.wsdl</EndPoint>
    </Component>
    <ConcreteProperty>
        <formula formulaId="formulaId0" type="type2" forChecking="true" diagnosisRequired="false"
                 threatDetectionRequired="false">....
         </formula>

        <formula formulaId="formulaId1" type="type2" forChecking="true" diagnosisRequired="false"
                 threatDetectionRequired="false">....
         </formula>
```

```
          .
          .
          .
     </ConcreteProperty>
</MonitoringConfiguration>
```

5.3.11. EvidenceAggregation Element

This section corresponds to section 5.1.1 in D2.2. There are no amendments to this part of the monitoring based certification model schema.

This element defines how often should the monitoring evidence being checked to create a new certificate, with new aggregated evidence (Figure 54).

In this element should be specified the:

- *The StartDate of the first aggregation,*
- *A choice of either:*
    - The *NumberOfEvents,* which is the number of the primitive monitoring events being aggregated, or
    - The *intervalsTime,* which specifies how often should the evidence being aggregated, and the intervalU*nit*, which declares the unit used to specify the intervals.
- *FunctionalAggregatorId*, which defines what type of aggregation should done in the events, and
- *IntermediateResults,* which is an optional 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.

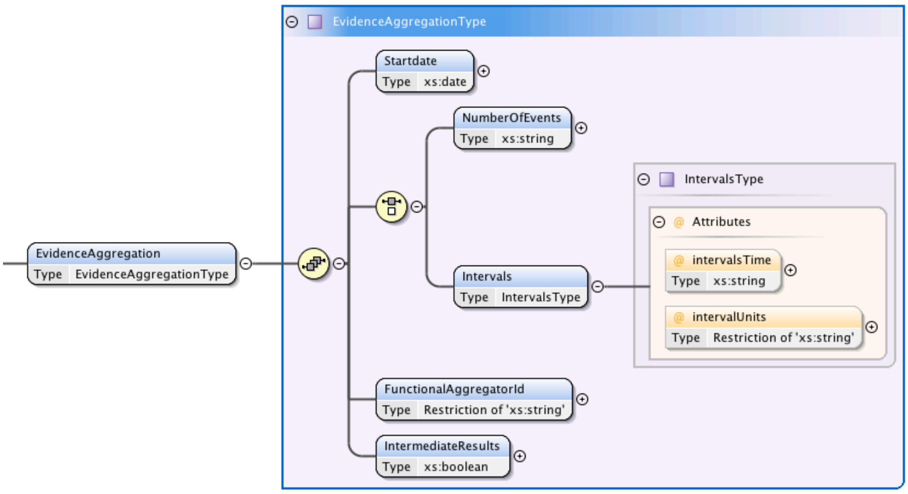


FIGURE 54 – MONITORING-BASED CM: EVIDENCE AGGREGATION TYPE

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 detailed evidence should be carried out at intervals of 720 hours in generating certificates, and should apply a Boolean value for the property.

```
<EvidenceAggregation>
          <StartDate>"2013-01-01"</StartDate>
          <Intervals intervalsTime="720" intervalUnit="hours"/>
          <FunctionalAggregatorId>Boolean</FunctionalAggregatorId>
          <IntermediateResults>True</IntermediateResults>
</EvidenceAggregation>
```

5.3.12. LifeCycleModel Element

This section amends section 5.1.8 in D2.2. In the new version of the schema for specifying monitoring based certification models, life cycle model elements are specified as instances of a new type of elements called StateTransitionModel (see section 5.3.6). This replaces the element type LifeCycleModelType in v1 of the schema.

This element defines all possible states that a certificate could have, as well as the transitions between the different states, and their conditions, which are references of predefined conditions of the certification model.

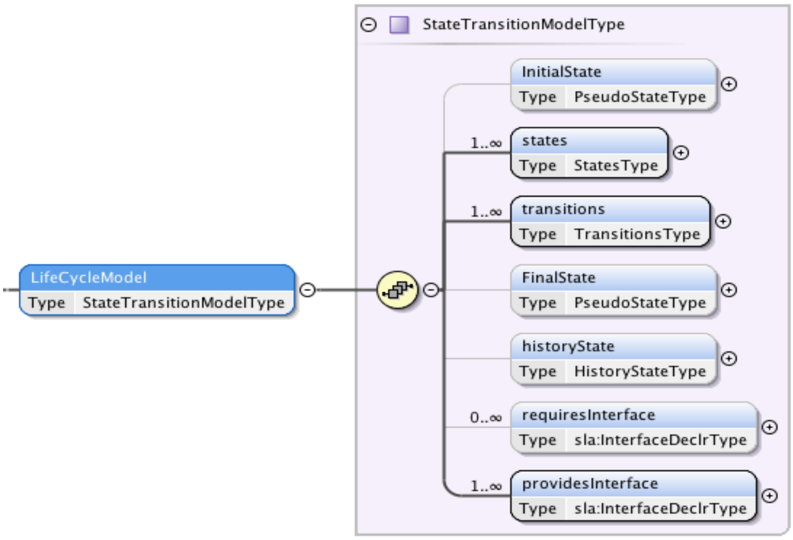


FIGURE 55 – MONITORING-BASED CM: LIFECYCLE MODEL TYPE

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.

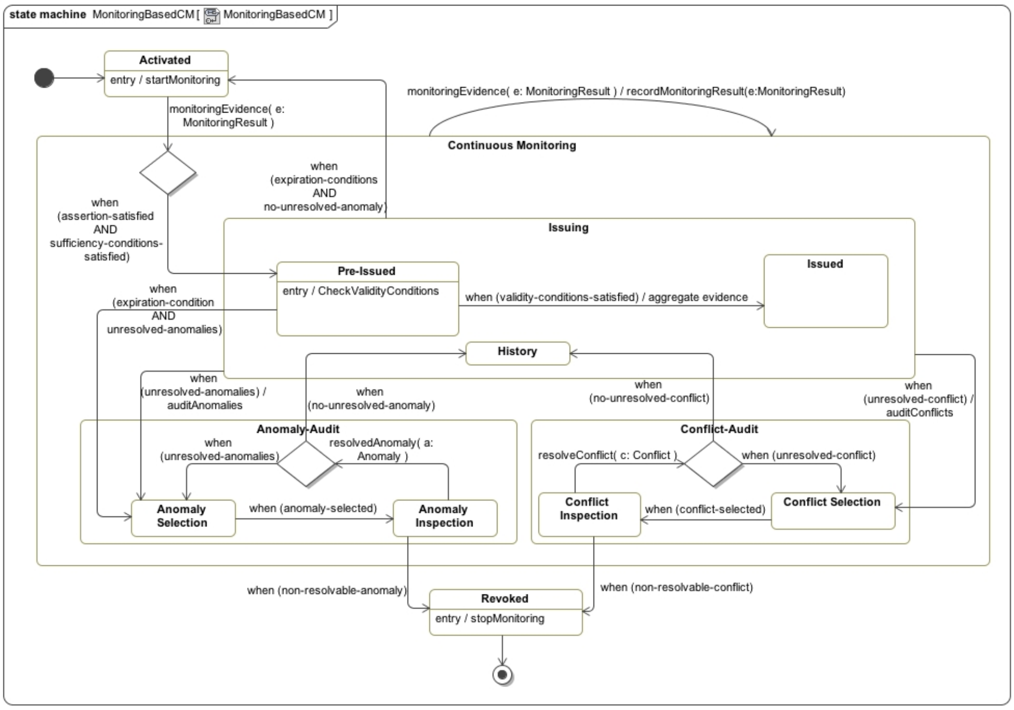

state machine MonitoringBasedCM [MonitoringBasedCM]
Activated
entry / startMonitoring
monitoringEvidence(e: MonitoringResult)
monitoringEvidence(e: MonitoringResult) / recordMonitoringResult(e:MonitoringResult)
Continuous Monitoring
when (expiration-conditions AND no-unresolved-anomaly)
when (assertion-satisfied AND sufficiency-conditions-satisfied)
Issuing
Pre-Issued
entry / CheckValidityConditions
when (validity-conditions-satisfied) / aggregate evidence
Issued
when (expiration-condition AND unresolved-anomalies)
History
when (unresolved-anomalies) / auditAnomalies
when (no-unresolved-anomaly)
when (no-unresolved-conflict)
when (unresolved-conflict) / auditConflicts
Anomaly-Audit
when (unresolved-anomalies)
resolvedAnomaly(a: Anomaly)
Anomaly Selection
when (anomaly-selected)
Anomaly Inspection
Conflict-Audit
resolveConflict(c: Conflict)
when (unresolved-conflict)
Conflict Inspection
when (conflict-selected)
Conflict Selection
when (non-resolvable-anomaly)
when (non-resolvable-conflict)
Revoked
entry / stopMonitoring

FIGURE 56 – MONITORING-BASED CM: UML DIAGRAM OF LIFE CYCLE MODEL

As shown in the figure, the life cycle model has an initial state called “Activated” and the states “Pre-Issued”, “Issued”, “Conflict Inspection”, “Conflict Selection”, “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”.

According to this model, the first state in the certificate’s lifecycle is called Activated, where the certificate is activated. After being activated, the certificate moves to the “ContinuousMonitoring” composite state, which consists of three other composite states named “Issuing”, “Conflict-Audit” and “Anomaly-Audit”. Whilst at this state (“Issuing”), the evidence required for the assessment of the certificate is continually gathered by the monitoring infrastructure. When the accumulated evidence, becomes sufficient according to the EvidenceSufficiencyConditions specified on the CM, the certificate moves to the state “Pre-Issued”, which is a sub-element of the composite state “Issuing”. 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”, from which it will generate a certificate instance and return it back to the requester.

Whilst a certificate type is at the “Issuing” state, the monitoring of the evidence continues and according to the *EvidenceAggregation* element, any additional operational evidence is recorded in an aggregated format. In case a warning is raised concerning a defined anomaly that is defined in the *Anomaly* element of the CM, the certificate moves to the sub state “Anomaly-Selection” of the composite state “Anomaly-Audit” (“when (no-unresolved-anomaly)” transition). In this state the framework will select the appropriate anomaly to be handled and the certificate will move to the “Anomaly-Inspection” state where the framework will try to resolve it. In case there are more anomalies to be resolved then the certificate will move back to the previous state (“Anomaly-Selection”) for another anomaly to be selected and when all anomalies are resolved, then the certificate moves to the “History” state, which is the state where it was prior to entering the “Anomaly-Audit” state, as signified by the guard condition of the transition. Otherwise, in case an anomaly cannot be resolved by the certification infrastructure, the certificate type moves to the “Revoke” state, which will lead to its termination.

Similar to the anomalies, in case a conflict occurs (according to the *Conflict* element of the CM), the certificate will move to the “Conflict-Audit” state, and more specifically to the “Conflict Selection” sub state (“when (unresolved-conflict) / auditConflicts” transition). At this state the framework will select the appropriate conflict to be handled and the certificate will move to the “Conflict Inspection” state. If the conflict is being resolved then the certificate will move to the “History” state. If more conflicts need to be handled, then it will move back to the “Conflict Selection” until all conflicts are resolved before moving to the “History” state. Otherwise, if a conflict cannot be resolved then the certificate will move to the “Revoke” state and it will be terminated.

Finally, when the expiration date of the certificate is reached as stated in the *ExpirationCondition* of the CM, and if there are no unresolved anomalies, the certificate will move back to “Activated” state (as depicted by the transition “when (expiration-conditions AND no-unresolved-anomaly)”). At this point the monitoring process will continue until sufficient evidence is available again for issuing another instance of the same certificate. Subsequently, if *SufficientEvidenceCondition* is satisfied, the certificate will move to the “Pre-Issued” state. However, if there are unresolved anomalies during the expiration of the certificate, it will first move to the “Anomaly-Audit”, as indicated by the transition “when (expiration-condition AND unresolved-anomalies)” in order to resolve any occurred anomaly, and then by moving back to the “Issued” state, from where the “when (expiration-conditions AND no-unresolved-anomaly)” condition will be triggered for the renewal of the certificate.

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 and 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>
<states>
 <state>
  <atomicState stateId="state1" name="Activated" description="Initial State"/>
 </state>
 <state>
  <compositeState stateId="compstate1" name="ContinuousMonitoring">
   <substate>
    <states>
     <state>
      <compositeState stateId="compstate2" name="Issuing">
       <substate>
        <states>
         <state>
          <atomicState stateId="state2" name="Pre-Issued"/>
         </state>
         <state>
          <atomicState stateId="state3" name="Issued"/>
         </state>
        </states>
        <transitions>
         <transition From="state2" To="state3">
          <WhenCondition negated="true">
           <LogicalExpression negated="true">
            <EvidenceAggregation/>
           </LogicalExpression>
          </WhenCondition>
         </transition>
        </transitions>
       </substate>
      </compositeState>
     </state>
     <state>
      <compositeState stateId="compstate3" name="Anomaly-Audit">
       <substate>
        <states>
         <state>
          <atomicState stateId="state4" name="AnomalySelection"/>
         </state>
         <state>
          <atomicState stateId="state5" name="AnomalyInspection"/>
         </state>
        </states>
        <transitions>
         <transition From="state4" To="state5">
          <GuardCondition negated="true">
           <LogicalExpression negated="true">
            <WhenAnomalySelected/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
         <transition From="state5" To="hstate1">
          <GuardCondition negated="true">
           <LogicalExpression>
            <WhenAnomalyResolved/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
         <transition From="state5" To="state4">
          <GuardCondition negated="false">
           <LogicalExpression>
            <UnresolvedAnoly/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
         <transition From="state5" To="state8">
          <GuardCondition>
           <LogicalExpression>
            <WhenUnresolvedAnomaly/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
        </transitions>
       </substate>
      </compositeState>
     </state>
     <state>
      <compositeState stateId="compstate4">
       <substate>
        <states>
         <state>
          <atomicState stateId="state6" name="ConflictSelection"/>
         </state>
         <state>
          <atomicState stateId="state7" name="ConflictInspection"/>
         </state>
        </states>
        <transitions>
         <transition From="state6" To="state7">
          <GuardCondition negated="true">
           <LogicalExpression>
            <WhenConflictSelected/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
         <transition From="state7" To="hstate1">
          <GuardCondition negated="true">
           <WhenConflictResolved/>
          </GuardCondition>
         </transition>
         <transition From="state7" To="state6">
          <GuardCondition negated="false">
           <LogicalExpression>
            <UnresolvedConflict/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
         <transition From="state7" To="state8">
          <GuardCondition>
           <LogicalExpression>
            <WhenUnresolvedConflict/>
           </LogicalExpression>
          </GuardCondition>
         </transition>
        </transitions>
       </substate>
      </compositeState>
     </state>
    </states>
    <transitions>
     <transition From="compstate2" To="state4">
      <WhenCondition>
       <Condition>
        <Anomaly>
         <Anomalies Id="2101"/>
        </Anomaly>
       </Condition>
      </WhenCondition>
     </transition>
     <transition From="compstate2" To="state6">
      <WhenCondition>
       <Condition>
        <conflictCondition>
         <Conflict Id="1100"/>
        </conflictCondition>
       </Condition>
      </WhenCondition>
     </transition>
     <transition From="compstate2" To="state1">
      <WhenCondition>
       <Condition>
        <expirationCondition>
         <ExpirationCondition Id="987"/>
        </expirationCondition>
       </Condition>
      </WhenCondition>
     </transition><transition From="compstate2" To="state4">
      <WhenCondition>
       <Condition>
        <expirationCondition>
         <ExpirationCondition Id="987"/>
        </expirationCondition>
       </Condition>
```

```
        <Condition>
         <Anomaly>
          <Anomalies Id="2101"/>
         </Anomaly>
        </Condition>
       </WhenCondition></transition>
     </transitions>
    </substate>
   </compositeState>
  </state>
  <state>
   <atomicState stateId="state8" name="Revoked"/>
  </state>
 </states>
 <historyState stateId="hstate1" refersToStateId="compstate2"/>
 <transitions>
  <transition From="state1" To="state2">
   <WhenCondition>
    <Condition>
     <evidenceSufficiencyCondition>
      <EvidenceSufficiencyCondition Id="1011"/>
     </evidenceSufficiencyCondition>
    </Condition>
   </WhenCondition>
  </transition>
 </transitions>
</LifeCycleModel>
```

6. TC-based Certification Model

This section amends section 6 in D2.2. The amendment made to the TC-based CM is the addition of a new model of TC support for certification.

As it was already explained in D2.2, the main innovation of CUMULUS project with regard to Trusted Computing (TC) based certification will be to bridge the gap that exists between software certification and hardware certification, in order to provide a comprehensive solution for system certification using hardware technology.

We consider two main scenarios for TC-based certification. In the first scenario, the TC certification model is not used as an independent model but it is rather leveraged to provide the trust that is needed for the validation of the Monitoring and Test Based certification. This means that the TC is not employed to directly certify a security property but instead used to increase the trust in other types of certifications. For instance, in a test-based certificate it can be used to prove that the platform configuration at runtime is the same as the one used for testing in a pre-production environment. In particular, the following section explains how TC is applied in order to guarantee that the Testing and Monitoring Modules running on the cloud can be trusted.

In the second scenario, the TC certification model will be used as an independent model to certify either platforms or services that are running in cloud. We would like to remark that this second use case would generalize the methodology already described for the first scenario in order to be able to cover more heterogeneous platforms and services, instead of only the monitoring and testing agents that CUMULUS intends to use.

In Section 6.1 we recall the TC-based CM as presented in D2.2, while in Section 6.2 we present a new model of TC support for certification. This new work focuses on the first scenario, which we understand as crucial for the reliability and acceptance of evidences gathered by monitoring and testing agents. We leave for deliverable D2.4 the presentation and details of the final (refined) standalone TC-based certification model.

6.1. TC-based Certification Model XML Schema Description

This section corresponds to Section 6.1 in D2.2. There are no amendments to the XML schema of the TC-based CM.

The initial TC certification model was proposed as follows (see Figure 57 and Table 6).

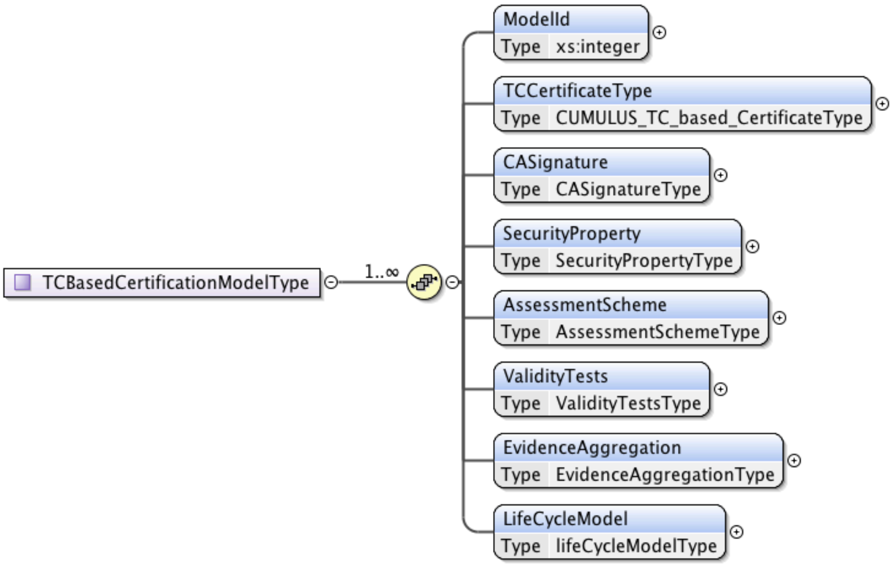


FIGURE 57 – TC CERTIFICATION MODEL SCHEMA

Certification Model field	**Type**	**Type specification**
Model_Id	integer	N/A

TCCertificateType	CUMULUS_TC_based_CertifiicateType	Type1 = Standard_Attestation_Certificate
		Type2 = Standard_Signature_Certificate
		Type 3 = Semantic_Signature_Certificate
		Certificate_Optional_Parameter;

CASignature	CASignatureType	name=”signatureType” type=String
		name="signer" type=String
		name="signature" type=String

SecurityProperty	SecurityPropertyType	name="Property_Category" type="xs:string" complexType name="AssertionType"

AssessmentScheme	AssessmentSchemeType	name="EvidenceSufficiencyCondition" type="EvidenceSufficiencyConditionType
		name="ExpirationCondition" type="ExpirationConditionType"
		name="Conflict" type="ConflictType"

ValidityTests	ValidityTestsType	External pre-condition: Parent of key loaded into TC module receiver. External pre-condition: Activated TC module on receiver.
EvidenceAggregation	EvidenceAggregationType	AggregatedResultsInfo EventSummary AggregatedValue IntermediateResults

LifeCycleModel	LifeCycleModelType	name="InitialState" type="IndividualStateType"
		name="transitions" type="TransitionsType"
		name="FinalState" type="IndividualStateType"

Table 6 – TC Certification Model Schema

In the following sub-sections we will describe the elements of the schema.

6.1.1. Model_Id Element

This section corresponds to section 6.1.1 in D2.2. There are no amendments to this part.

Model_Id is the unique identifier of the certification model instance, compliant with the certification meta-model in section 2.5.

6.1.2. TCCertificateType Element

This section corresponds to section 6.1.2 in D2.2. There are no amendments to this part.

TCCertificateType represents different kinds of TC-based Certification. For the type specification see alsoTable 6.

6.1.3. CASignature Element

This section corresponds to section 6.1.3 in D2.2. There are no amendments to this part.

CASignature represents the signature of the certification authority that has defined the certification model. The element is defined in accordance to the CUMULUS meta-model.

6.1.4. Security Property

This section corresponds to section 6.1.4 in D2.2. There are no amendments to this part.

SecurityProperty is the element in the schema that defines the security property that is to be certified by the particular instance of the Certification Model. The format of this element is compliant with the one described in Section 2.3.

6.1.5. AssessmentScheme Element

This section corresponds to section 6.1.5 in D2.2. There are no amendments to this part.

This element follows the same structure as the one defined in D2.2.

6.1.6. ValidityTests Element

This section corresponds to section 6.1.6 in D2.2. There are no amendments to this part.

A certification model may, in addition to the assessment scheme, define extra validity tests as preconditions for issuing a certificate of a given type.

This element follows the same structure as the one described in D2.2.

6.1.7. EvidenceAggregation Element

This section corresponds to section 6.1.7 in D2.2. There are no amendments to this part.

This element follows the same structure as the one described in D2.2.

6.1.8. LifeCycleModel Element

This section corresponds to section 6.1.8 in D2.2. There are no amendments to this part.

This element follows the same structure as the one described in D2.2, and is defined in accordance to the certification meta-model life-cycle element in section 2.5.

6.2. TC Support for Certification

This section has no corresponding section in D2.2. It describes a new model for providing TC support for certifiction models.

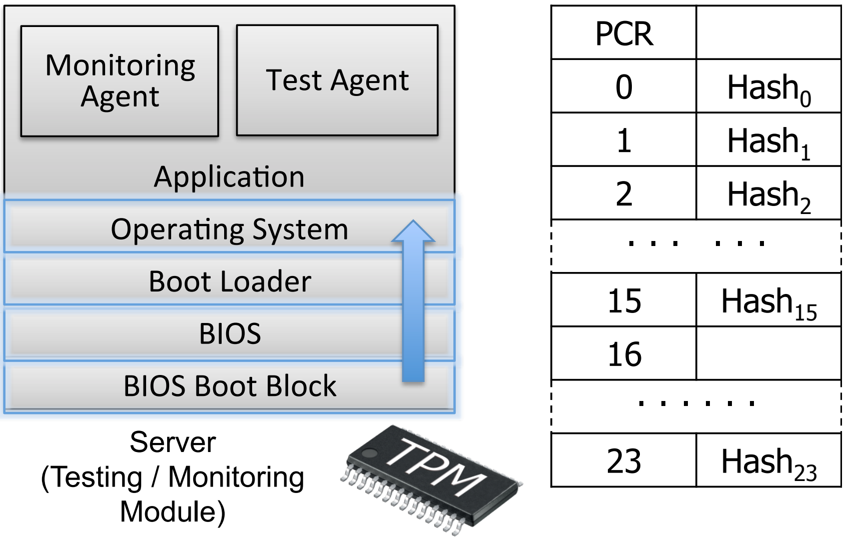
In this section, we present a model for the use of TC in order to guarantee that the Testing and Monitoring Modules deployed and running on the cloud are fully trusted. We will first introduce the basic concepts related to TC support for certification, and how these relate to the schema of the TC support model.

6.2.1. TC Support Basics

This section has no corresponding section in D2.2.

Once the testing or monitoring agent is deployed on a cloud platform, in order to support software agent integrity we need to bind the state of the agent in question to a state of the underlying platform. To support the binding process we need to include TC-specific artefacts in the certification model schema. The TC-specific artefacts and binding process are shown in Figure 58.

The TC binding process is based on the TPM technology. We will describe in details the concepts and specific fields of TPM for a better understanding of the binding process and the defined TC support schema.



PCR	
0	Hash_0
1	Hash_1
2	Hash_2
. . .	. . .
15	Hash_{15}
16	
. . . .	
23	Hash_{23}

FIGURE 58 – BINDING OF TESTING/MONITORING AGENT TO A PLATFORM STATE

TPM v1.2 chipsets have 24 Platform Configuration Registers (PCRs). The PCRs are series of 20-byte registers that are used to store system measurements. This is the length since 20 bytes is the length of the output of the SHA-1 algorithm[7], and in most of the cases this operation is used to compute the checksum of a component of the system. The PCRs are reset to a known value on every boot and they cannot be overwritten[8]. The only way to add data to a PCR is with the *Extend* operation that combines the previous value of a PCR with 20 new bytes of data using the SHA-1 algorithm. In that way, each new state stored in a PCR is dependent on the value of the previous state. Thus, the PCRs can keep track of unlimited number of measurements (*Extend operation*) and each PCR index has its own purpose[9], described as following:

- BIOS, ROM, Memory Block Register [PCR index 0-4]
- OS loaders [PCR index 5-7]
- Operating System (OS) [PCR index 8-15]
- Debug [PCR index 16]
- Localities, Trusted OS [PCR index 17-22]
- Applications specific [PCR index 23]

The fundamental concept of binding software agent to a system (platform) state is the use of TPM functionality of binding cryptographic keys to PCR values. This TPM functionality allows (administrative) entities to request to a TPM, via defined APIs, to generate a key pair and seal the private key to a set of PCRs' values. The TPM chip generates keys internally and the private key, of a public/private key pair, is bound to the selected set of PCRs. In that way, all operations using the private key of the key pair are performed inside the TPM chip and only if the state of the PCRs has not changed (i.e., if the current values of the PCRs are the same as those when the key was bound to). For example, if a software agent (on a server platform) can sign a challenge message with the key this implies that the platform, to which the key is bound to, remains in the same state. Another important aspect of the key binding process is that the key bound to the TPM's PCRs can only be used on the same TPM chip (in the same platform). This key is of type of non-migratable keys of TPM[10].

Given the above functionality, we have the following two application binding possibilities:

- a software agent can be considered to be *running on a valid platform state*, if the agent can use a cryptographic key already bound to a valid platform state (a set of PCR values).
- a software agent can be considered to be *in a valid state* if the agent can use a cryptographic key already bound to both a valid platform state (a set of PCR values) and a valid software agent state (stored in a specific PCR).

6.2.2. TC Support Model

This section has no corresponding section in D2.2.

This section describes the defined model providing TC support to any of the CUMULUS CMs, and an explanation of the artefacts that compose it. The model is to be included (referenced) as an integral part of any CUMULUS CM along with the artefacts defined by the model. Figure 59 shows a graphical representation of the TC support model, while Appendix 10.5 shows the corresponding XML schema.

[7] In TPM v2.0 there is another PCR bank for SHA-256 with PCR length of 32 bytes.

[8] We mean that PCR values cannot be set to specific values, but they can only be extended with new values. PCRs 16 and 23 can only be reset (with zeros) if access to TPM is given for that.

[9] https://www.trustedcomputinggroup.org/files/static_page_files/E55A303C-1A4B-B294-D066E66A82DAE27D/TPM%20Main-Part%202%20TPM%20Structures_v1.2_rev116_01032011.pdf

[10] https://www.trustedcomputinggroup.org/files/static_page_files/72C26AB5-1A4B-B294-D002BC0B8C062FF6/TPM%20Main-Part%201%20Design%20Principles_v1.2_rev116_01032011.pdf

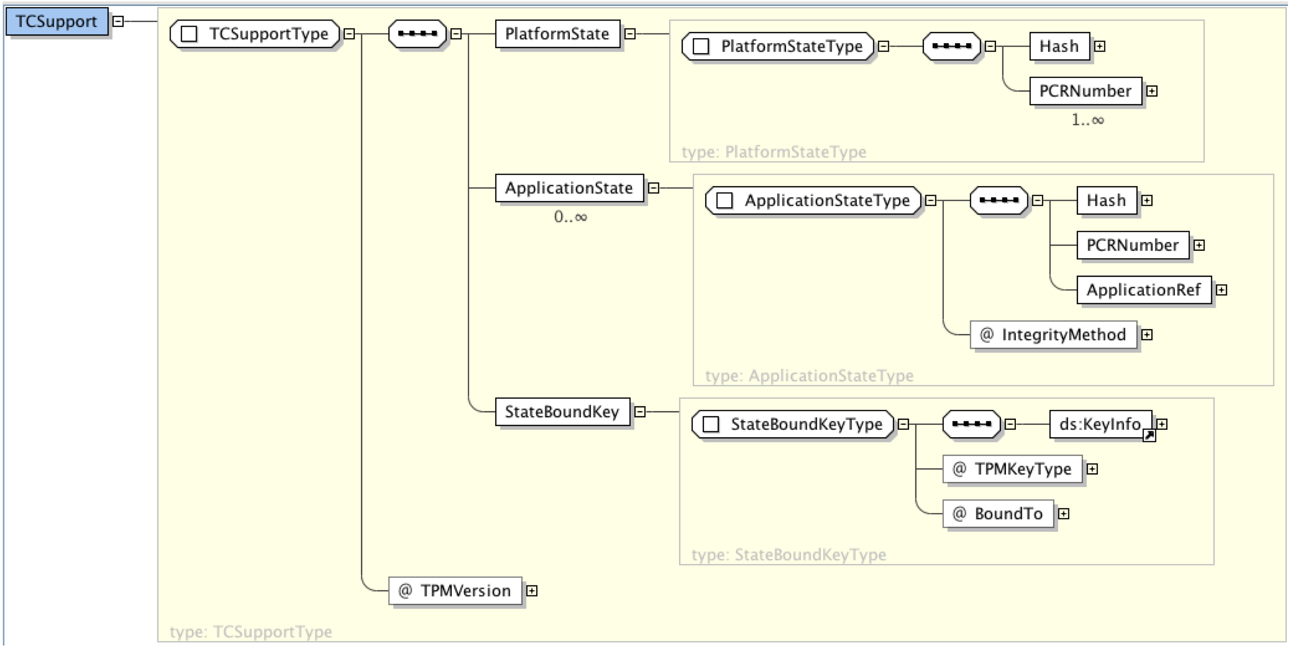


FIGURE 59 – TC-SUPPORT MODEL SCHEMA

The rationale of each field included in the TC-Support model schema is given bellow:

PlatformState

Specifies the state of the platform, it contains integrity measurements (checksums) of the platform stack from the BIOS up to the OS level. This field does not include the state of the high level applications.

PlatformState.PCRNumber

PCR indexes used to store the measurements of the platform. For example, one can use PCR indexes 0-15. A detailed explanation and rationale of which specific PCRs are used by the TC mechanisms will be given in deliverable D3.2.

PlatformState.Hash

This hash value is the result of the computation of the platform components' integrity values stored in the selected PCRs.

ApplicationState

It specifies the state of the application that must have its integrity ensured, in this case the Testing and Monitoring agent applications.

ApplicationState.ApplicationRef

Contains a list of the elements, application-specific files that compose the application whose integrity must be ensured. ***Important***: the list of application-specific files must be treated as an ***ordered list*** in order to facilitate the integrity verification of the overall application.

ApplicationState.IntegrityMethod

Defines the method used to compute the integrity of the overall application as a single value. A possible way of computing an application state is by iterating over the list of application-specific files in the given order, where the integrity of each file is computed as a function of the integrity value computed on the previous file in the list[11].

[11] See for example linked timestaping (http://en.wikipedia.org/wiki/Linked_timestamping)

ApplicationState.PCRNumber

The PCR index used to store the integrity checksum of the application. For example, one can use PCR index 23 to store the integrity checksum.

ApplicationState.Hash

It is the result of the integrity measurement of the trusted application. If some components of the application change the resulting value of the checksum will be different from this one.

StateBoundKey

The cryptographic key stored in the TPM that is bound to some PCR values. This means that if any of the values of the associated PCRs have been modified, the key will not be accessible.

StateBoundKey.KeyInfo

It contains, among more general information, the public portion of the cryptographic key, including its crypto schema type (RSA, DSA, etc). It is important to note that the KeyInfo element is defined as an element referencing <ds:KeyInfo> element of the XML digital signature schema[12], and as such a more flexible key representation can be used, for example a X.509 public key certificate, which as a consequence provides a more convenient way of handling key management at application-level.

StateBoundKey.BoundTo

Specifies two options: *(i)* if the key is bound to *a valid platform state;* or *(ii)* if the key is bound to both, *a valid platform state* and *a valid software agent state*. This information is to be used by application-level operations to determine what attestation verification is to be undertaken/accepted or not by the CUMULUS platform.

In the following we describe some possible (application-specific) interpretations of the BoundTo element values. According to the values we can have the following interpretations:

1. *Restrictive*: refers to option *(ii)* above. The key can only be used if the states of the platform and of the application remain the same as at the time of key creation (certification).

   Pros: It is not necessary to perform remote attestation on the integrity of the software agent because such attestation is implicit by the use (binding) of the key.

   Cons: Every time the application (agent) changes (updates) it is required to re-certify the integrity of the application state, and create a new key bound to this new state.

2. *Permisive*: refers to option *(i)* above. Since the key is bound only to a valid platform state, a valid application state must be referenced from a trusted source of information (e.g., internal CUMULUS database with valid states of (CUMULUS) software components).

   Pros: In case of updates on the application level, it is not necessary to re-certify the integrity of the new application state but only to re-evaluate and update the trusted source of information with the new state.

   Cons: It requires periodical remote attestations to ensure the authenticity and integrity of the application (CUMULUS components) current state.

StateBoundKey.TPMKeyType

Specifies if the key can be used out of the TPM (*migratable key*) or can only be used in the same TPM where the key was created (*non-migratable key*). For the current implementation we have restricted this value to non-migratable key.

[12] http://www.w3.org/TR/xmldsig-core/#sec-KeyInfo

An illustrative example of the TC-Support for Certification is given in Figure 60:

```
<ns3:TCSupport TPMVersion="1.2"
   xmlns:ns2="http://www.w3.org/2000/09/xmldsig#" xmlns:ns3="urn:cumulus:tcsupport">
   <PlatformState>
      <Hash
AlgRef="http://www.w3.org/2000/09/xmldsig#sha1">LSymIVER7tScbbjZAENunWQhDGM=</Hash>
      <PCRNumber>0</PCRNumber>
      <PCRNumber>1</PCRNumber>
      <PCRNumber>2</PCRNumber>
      ...
   </PlatformState>
   <ApplicationState IntegrityMethod="urn:cumulus:tcsupport:integritymethod:linked-timestamping">
      <Hash
AlgRef="http://www.w3.org/2000/09/xmldsig#sha1">rU6BBxN4h/0HqLMCNhJj7ZS3yMw=</Hash>
      <PCRNumber>23</PCRNumber>
      <ApplicationRef>
         <ElementRef>cumulus:monitoringcomponent:monitor.jar</ElementRef>
         <ElementRef>cumulus:monitoringcomponent:config:config.xml</ElementRef>
         <ElementRef>cumulus:monitoringcomponent:lib:EventManager.jar</ElementRef>
         ...
      </ApplicationRef>
   </ApplicationState>
   <StateBoundKey BoundTo="PlatformAndApplicationState" TPMKeyType="Non-MigratableKey">
      <ns2:KeyInfo>
         <ns2:KeyValue>
            <ns2:RSAKeyValue>

<ns2:Modulus>6h6uowDi1q5LAEyD3ghOdZcS9+VlwFeFwu+C9z4MRyunpeFKl0nZ2qtE97LoxHfKB

           a+LJsRGbLOeGxZc9w3me0VZzQJ8LsrIrbG+Mvtk4eZkEQrF02tpC/zIMe30T4B0kpYkI91elpeMp/n1R
               WzUH8+a/5cWVUnHT80=</ns2:Modulus>
               <ns2:Exponent>AQAB</ns2:Exponent>
            </ns2:RSAKeyValue>
         </ns2:KeyValue>
      </ns2:KeyInfo>
   </StateBoundKey>
</ns3:TCSupport>
```

FIGURE 60 – TC-SUPPORT FOR CERTIFICATION EXAMPLE

As we can see in the example, the platform state is calculated with the SHA-1 algorithm (<Hash AlgRef="http://www.w3.org/2000/09/xmldsig#sha1">), the input of the algorithm is the content of the PCRs 0, 1, 2... (<PCRNumber>). The application state is also obtained with the same algorithm, the hash is iteratively obtained by calculating the SHA-1 of the elements that compose the application (<ApplicationRef> <ElementRef>cumulus:monitoringcomponent:monitor.jar</ElementRef> …).

The key is non-migratable and it is bound to the platform state and to the software agent state (<StateBoundKey BoundTo="PlatformAndApplicationState" TPMKeyType="Non-MigratableKey">). It is a RSA key and the public portion is contained in the modulus artefact (<ns2:Modulus>).

7. Advanced Certification Models

This section is new. There is no corresponding section in D2.2.

According to the Description of Work, the Incremental, Hybrid and Multi-layer versions of the Certification Models are due in the last version of the deliverable (CUMULUS Consortium, D2.4 Final Certification Models, 2015), nonetheless we give here a first overview of our approach.

7.1. Multi-layer certification

CUMULUS aims to provide a certification approach that addresses the multi-layer structure of cloud environments. The cloud paradigm in fact offers a powerful approach to the provision of infrastructure, platform, and software services that, on one side, increases performance, flexibility, and effectiveness, while, on the other side, raises significant concerns regarding security at each cloud stack layer. In this context, CUMULUS is developing an integrated framework of models and processes that support the certification of security properties that insist on multiple levels of the cloud stack. Our multi-layer approach is based on a single CM that specifies the security property to be certified on a given ToC involving different levels of the cloud stack (e.g., SaaS-PaaS or SaaS-IaaS, SaaS-PaaS-IaaS).

7.1.1. Multi-layer Test-based certification

Focusing on test-based CMs, to accomplish multi-layer requirements, we refined the ToC perimeter using ToTs (see Section 4.1.3). ToTs refer to mechanisms to be tested at different cloud layers to prove a given property on a given ToC. The testing process models the interconnection between the mechanisms at different layers allowing integration testing. The testing activities are performed by test-based collectors instantiated with a specific configuration and different ToTs (see Section 4.1.3). The collector configuration is tailored to the layer where the corresponding mechanism to be tested is deployed (e.g., it runs on a specific cloud stack for testing confidentiality at rest, it runs on external and intercloud facilities for testing confidentiality in transit).

A complete approach to multi-layer certification can consider two main scenarios: multi-layer certification from scratch and incremental multi-layer certification. In the following, we briefly summarize the two scenarios and discuss a test-based example of multi-layer certification from scratch.

1. Multi-layer certification from scratch assumes a scenario where all ToTs specified in the CM refers to security mechanisms that are not certified. The certification authority starts a complete certification process (similar to the one used for single-layer certification) evaluating (by monitoring or testing) relevant security mechanisms with respect to corresponding ToTs, to the aim of certifying the security property target of the multi-layer certification. As an example, let us consider a cloud service provider that wants to certify its service for property data leakage prevention. Data leakage prevention requires a multi-layer certification, where SaaS communications must be encrypted and IaaS data must be stored in an encrypted storage. In other words, the multi-layer certification must first evaluate the mechanism implementing encrypted communication and then evaluate the mechanism implementing the encrypted storage. We note that this is not the only possible security configuration supporting data leakage prevention. As an example, a different CM can specify an access control mechanism instead of an encrypted storage.
2. Incremental multi-layer certification mimics the incremental certification process in Section 7.1 to provide an efficient multi-layer certification approach. In particular, like for incremental certification, we aim at reusing certificates of single mechanisms involved in the multi-layer certification to reduce the amount of certification requested in the CM. Certificates of single mechanisms are then used to manage requirements on different layers of the cloud stack that

> could affect the multi-layer certification process. More in detail, results stored in certificates of single mechanisms are matched against requirements stated in the ToT of the CM that refers to these mechanisms. If they are compatible, there is no need for testing/monitoring the security mechanism under consideration[13]. As a consequence we have three possible scenarios. Let us again consider data leakage prevention and the two mechanisms (communication encryption for SaaS and encrypted storage for IaaS, respectively) at point 1. In the first scenario both mechanisms have a certificate that matches requirements in the corresponding ToTs in the CM. In this case, no additional certification is required. In the second case, one certificate matches requirements in the corresponding ToT, while the second one matches only a subset of requirements in the corresponding ToT. In this case, an incremental certification of the second mechanism is needed. In the third scenario, both certificates violate the requirements in the corresponding ToTs. In this case a multi-layer certification from scratch is needed.

Below we present a detailed example of test-based multi-layer certification from scratch considering property data leakage prevention and mechanisms supporting encryption of communication channel and encrypted storage.

Let us consider a service s that aims at certifying property data leakage prevention in a given cloud stack. Service s needs a multi-layer Certification Model where the Target of Certification is both SaaS and IaaS. In fact, s must first guarantee confidentiality of data in transit through a secure channel implemented using an encryption mechanism (SaaS layer). We note that property data leakage prevention could also require to secure (if necessary) the internal channel from/to the server platform (PaaS layer) again with an encryption mechanism. For simplicity, here, we assume a direct channel between SaaS and IaaS. Also, data leakage prevention requires an encrypted storage to guarantee the confidentiality of data at rest (IaaS layer).

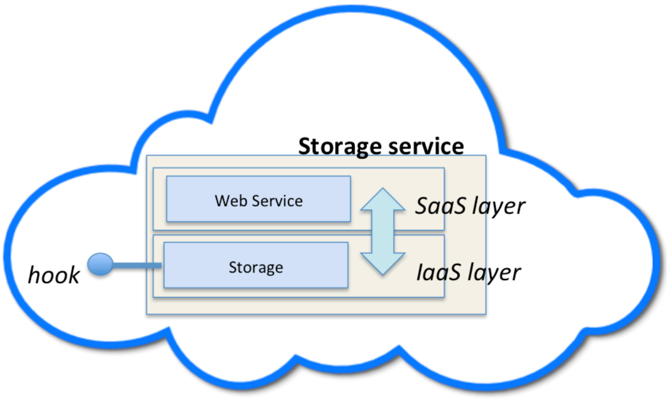


FIGURE 61 – STORAGE SERVICE TO BE CERTIFIED

As already discussed, we define a single Certification Model with a single security property, using different mechanisms (encrypted channel and encrypted storage) that insist on different layers (SaaS and IaaS).
The description of requirements on single mechanisms and corresponding test cases to be executed in the multi-layer certification process can be specified in two different Abstract Collectors connected to two ToTs. The first Abstract Collector describes the test cases to be executed against services (ToT) implementing the secure channel. The abstract collector defines both test cases to be executed in a laboratory environment and dynamic test cases to be executed only when the service is put in production. Laboratory test cases can include both "static code review", where the code is formally examined and output is a passed/not passed check list, and more common functional test cases examining the correct behaviour of the security mechanism. Dynamic test cases include conditions triggering their execution, that

[13] In some cases an integration test is required anyway, but all the independent tests are still valid, and the integration testing can be obtained incrementally.

is, a “model control flow” that examine the functioning of the service, for example the real traffic received and generated by it in a give time window.

The XML code that describes an example of the first abstract collector and corresponding ToTs of the multi-layer CM is the following:

```
[...]
  <Toc Id="ToC1">
    <CloudLayer>SaaS </CloudLayer>
   <CloudLayer>IaaS </CloudLayer>
    <ConcreteToc>StorageService</ConcreteToc>
    <TocDescription>Application</TocDescription>
    <TocURI>10.0.0.155</TocURI>
    <ToTs> [.....] </ToTs>
    <OperativeCondition>
      <TocTechnicalSpecifications>
        <TocVendor>XYZ</TocVendor>
        <TocRelease>1.0</TocRelease>
        <TocDate>2014-09-24</TocDate>
      </TocTechnicalSpecifications>
    </OperativeCondition>
  </Toc>
[...]
<SecurityProperty SecurityPropertyId="Id102" SecurityPropertyDefinition=".....">
    <sProperty class="http://cumulus-project.eu/security-properties#DSI:data-leakage-control:data-leakage-
prevention">
[....]
<--- This Abstract Collector describes test cases to be executed against service s (ToC) implementing the secure
channel (test cases in laboratory environment and in dynamic environment) ---->
   <Collectors>
        <AbstractCollector Id="abstractCollector1">
          <Aggregator >
            <ModelLink>http://www.cumulus-project.eu/Model55.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            [....]
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <TestType>Static code review</TestType>
          <TestDescription> Static code review/testing</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model55.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>1</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
          <TestCase>
          <ID>checkList#1</ID>
          <Description>https support check</Description>
           <TestInstance Operation="readCode">
            <Preconditions/>
            <HiddenCommunications/>
            <Input></Input>
            <ExpectedOutput>true</ExpectedOutput>
```

```
                <PostConditions></PostConditions>
               </TestInstance>
               </TestCase>
              [... other test cases ...]
          </TestCases>
        </AbstractCollector>
[...]
      <Collector isStatic="true" Id="col1">
        <AbstractCollector>abstractCollector1</AbstractCollector>
      </Collector>
[...]
```

The second Abstract Collector describes the test cases for the encrypted storage (ToT), and includes both laboratory test cases and dynamic test cases done in production. In the latter case, the IaaS layer must provide a hook that allows CUMULUS framework to collect data on encrypted storage behaviour. As an example, test type “input partitioning” can be used, that is, a set of pairs input/expected output are used to test the IaaS service through the hook. In particular, the input is sent to the service and the hook is used to check directly on the file system whether the data chunk has been correctly encrypted (matching the expected output).

The XML code that describes an example of the second abstract collector and corresponding ToT of the multi-layer CM is the following:

```
[...]
    <AbstractCollector Id="abstractCollector2">
              <Aggregator >
                <ModelLink>http://www.cumulus-project.eu/Model56.html</ModelLink>
                <TestMetric>
                  <OperationCoverage>1</OperationCoverage>
                  <InputPartitionCoverage/>
                  <BranchCoverage/>
                  <ConditionCoverage/>
                  <PathCoverage/>
                  <AttackCoverage/>
                  <Other> </Other>
                </TestMetric>
                 [....]
              </Aggregator>
              <TestCategory>Functionality</TestCategory>
              <TestType>Model control flow</TestType>
              <TestDescription>Check service functioning</TestDescription>
              <TestGenerationModelLink>http://www.cumulus-project.eu/model56.html</TestGenerationModelLink>
              <TestAttributes>
                <TestAttribute>
                  <ID>1</ID>
                  <Name>cardinality</Name>
                  <Value>1</Value>
                </TestAttribute>
              </TestAttributes>
              <TestCases>
              <TestCase>
              <ID>1</ID>
              <Description>Encryption Data check in the storage</Description>
               <TestInstance Operation="readData">
                <Preconditions/>
                <HiddenCommunications/>
                <Input>DataIndex</Input>
                <ExpectedOutput>DataChunk[DataIndex]</ExpectedOutput>
                <PostConditions>Decryption(DataChunk[DataIndex]) = PlainText</PostConditions>
               </TestInstance>
               </TestCase>
```

```
                [....]
              </TestCases>
           </AbstractCollector>
[...]
      <Collector isStatic="true" Id="Collector3">
         <AbstractCollector>abstractCollector2</AbstractCollector>
      </Collector>
      <Collector isStatic="false " Id="Collector3">
     <ConditionForExecution>
           <Event><Action> </Action>
         <Condition> ConditionForExecution#2 </Condition>
              <Value>1</Value>
           </Event>
        </ ConditionForExecution >
         <AbstractCollector>abstractCollector2</AbstractCollector>
      </Collector>
   </Collectors>
[...]
```

In addition if required by the mechanisms or by the property to be certified, an additional abstract collector can be implemented to specify integration testing activities. In this example, integration testing is not considered because the two mechanisms are disjoint from a testing point of view. To conclude, the Abstract collectors are instantiated in testing collectors configured for executing real testing activities based on the ToTs, at different cloud layers.

Multi-layer certification of cloud services is a fundamental aspect of a cloud certification scheme. In the last year of the project, WP2 will study and refine the above approaches providing a consistent solution to the management of multi-layer certification, which also integrates with single-layer certification.

7.1.2. Multi-layer Monitoring-based certification

Monitoring-based certification is multi-layer if events come from different layers.

7.2. Incremental certification

Incremental certification models are used to manage changes at any layer of the cloud stack that could affect certified security properties without the need to (re-)certify artefacts from scratch. It covers all aspects related to the adaptation of an existing certificate through the possible reuse of a part of it. Incremental certification is provided by considering evolutions of services and their interactions, and by verifying the validity of previously verified properties following changes in the stack.

7.2.1. Incremental Test-based certification

In this section, we concentrate on the management of security certifications and corresponding issued certificates upon a change in software/service/cloud change. Uncontrolled changes may in fact invalidate the security certificate awarded to a given software/service/cloud. Traditional approaches to security certification (Damiani, Ardagna, & Ioini, Open source systems security certification, 2009) (Anisetti, Ardagna, & Damiani, Fine-grained modeling of web services for test-based security certification, 2011) usually do not deal with evolving software/services and prescribe re-certification of software/services upon the delivery of new versions. This approach results in high certification costs and overheads, since each change to the service triggers a re-certification process from scratch, that is, for test-based scenario, offline/static testing in a pre-production environment. Recertification from scratch is especially critical in the cloud where services are subject to continuous changes, as for instance the one dictated by cloud migration functionalities. For example in a SaaS scenario, apply a re-certification process from scratch for each service evolution is not an option, and the need of services that adapt to changing conditions is related to the definition of cloud computing.

A first approach sketched in this section aims to address the above problem, by providing a solution for the management of incremental certification of evolving software/services. Our solution responds to the need of providing a low-cost certification scheme for evolving and cloud services, which does not prescribe re-certification from scratch. We therefore define a preliminary scheme that re-uses existing certificates and related evidence limiting the amount of new and additional certification activities. Our solution concentrates on both test-based and monitoring-based evidence, and aims to minimize test generation and monitoring activities for certifying evolving services.

An incremental certification process renews a certificate by re-using, as much as possible, the certification evidence available from older certificates. An incremental certification process can reduce the costs and overheads of the certification of evolving software services at any level of the stack, while providing a similar level of assurance.

Our approach to the management of the incremental certification can be classified depending on the required level of re-certification as follows.

- Certificate adaptation: an event, such as a change of the Target of Certification, occurs but there is no need of re-certification, because there is no code modification and all the targets of tests are not changed; in this case, testing activities (dynamic tests in a test-based scenario or monitoring tests) still run and the certificate remains in the ISSUED state. For example, this happens when a Web Service is moved within the Cloud.
- Partial re-certification: upon the release of a new version of a certified software service at SaaS, PaaS or IaaS level, some of the Targets of Tests change, some are not valid anymore, others are added in the new version. In this case, an incremental certification process is triggered: in a test-based scenario the Certificate goes to the SUSPENDED state and only a portion of the test-based evidence in the original certificate needs to be re-executed or substituted by new test cases. Dynamic testing can then be used to verify that unchanged parts have not been affected by changes in other software/services and behave as expected.
- Full re-certification: when there is a new version of a certified software service at SaaS, PaaS or IaaS level, the targets of tests change, and all the dynamic tests fail; in a test-based scenario the certificate goes to the REVOKED state, a new certification process is applied from scratch and restarts from the static/offline scenario.

Below, we illustrate the working of the above classes.

Certificate adaptation

A certificate adaptation typically occurs when some events, which could limit the validity of the certificate, happen, but the dynamic certification process can go on since there are no significant changes in the Targets of Tests. For example, an e-booking Web Service is moved within the Cloud (internal migration). This operation affects the end points in our Certification Model and triggers an adaptation process.

In such a situation the Web Service code is the same, the Targets of Tests are not modified, the dynamic tests still run and give the same level of assurance; in other words there is no need to reissue the certificate or modify it, and the certificate remain in the ISSUED state.

We note that in case the original environment is restored (in our example the e-booking Web Service is moved back to the first Cloud Provider), a new adaptation process is triggered.

Partial recertification

A partial re-certification is triggered by the release of a new version s' of a certified service s, and aims at re-using existing evidence included in the certificate awarded to s, to re-issue a new certificate that is valid for s'.

Static/offline recertification can be necessary only for some small changed parts. The two sets of Targets of Tests, as defined in the Target of Certifications of s and s', are used to manage partial re-certification activities, including the specification and execution of additional test cases and/or the re-execution of existing ones. If we define a set of Targets of Tests as *test model*, the first step in partial re-certification compares the two test models to identify the parts of the models, which have been affected by service changes.

We consider how partial re-certification is handled distinguishing between: i) test model extension, ii) test model reduction, and iii) test model update.

For example, in a test-based scenario, a new version of an e-booking Web Service is released with new functionalities: some old APIs are deprecated and some Targets of Tests change. In this situation, there is the need to (re-)run some static tests. The Certificate then goes to the SUSPENDED state and only a portion of the Test-based evidence in the original certificate needs to be re-executed or substituted by new test cases. Once the tests are updated and run and if the new level of assurance is sufficient, the Certificate goes back to the ISSUED state.

In case the original environment is restored, that is it is restored the old version of the e-booking Web Service, the same partial re-certification process takes place again.

Also, dynamic testing can be used to verify that unchanged parts have not been affected by changes in the new version of a software/service and behave as expected.

Full recertification

A full re-certification is triggered by a software/service update that affects all the targets of tests, making all test cases invalid. A new certification process should be applied from scratch.

The full re-certification also applies when the certificate is either revoked (evidence goes below threshold) or expired, as already defined in the Certification Life Cycle.

A particular case of full re-certification is when a certificate refers to two or three different Cloud layers, for example SaaS and IaaS, such a Web Service and its corresponding Virtual Machine. When the IaaS component changes (for example, the Virtual Machine is moved from a Cloud Provider to another one) the full process must restart from the static/offline case.

7.2.2. Incremental Monitoring-based certification

Monitoring-based certification is incremental by definition.

7.3. Hybrid certification

This section has no corresponding section in D2.2. It describes a new category of advanced Certification Models, which is the Hybrid Certification Models, that cross-check evidence gathered from Testing and Monitoring.

The key concept underpinning a hybrid certification model is to cross-check evidence regarding a security property that has been gathered from testing and monitoring and, provided that there is no conflict within it, to combine it providing assurance for properties. Consider, for example, a scenario where the property to be certified is cloud service availability. If availability is measured as the percentage of the calls to service operations for which a response was produced with a given time period d, a monitoring check should verify exactly this condition. However, the trace of service calls that has been examined by the monitoring process might not cover all the operations in the service interface or the expected peak workload periods of the underlying infrastructure. In such cases, before issuing a certificate for service availability, it would be necessary to test any of the above service usage conditions that have not been covered yet. The combination of monitoring and testing can be attempted in two basic modes:

(1) The dependent mode – In this mode, a security property is assessed for a TOC by a primary form of assessment (monitoring or testing) which triggers the other (subordinate) form in order to confirm and/or complete the evidence required for the assessment.

(2) The independent mode – In this mode, a security property is assessed for a TOC by both monitoring and testing independently without any of these assessments being triggered by outcomes of the other. Then at specific points defined by the evidence sufficiency conditions of the certification model the two bodies of evidence are correlated and cross-checked to complete the hybrid assessment.

Beyond the elements of certification models that were overviewed in III.A, a hybrid certification model should also define: (a) the mode of hybrid certification; (b) the way of correlating monitoring and testing evidence; (c) conditions for characterising these types of evidence as conflicting, and (d) the way in which a final overall assessment of the property can be generated based on both types of evidence.

In the following, we give examples of hybrid certification models of both modes, formalise them in EC-Assertion and use this formalisation to examine generic relationships that exist in hybrid models.

Example 1: Hybrid, dependent mode models

Our first example shows the use of a hybrid approach in certifying data integrity-at-rest. As defined in (Consortium, 2013), this property expresses the ability to detect and report any alteration of stored data in a target of certification (TOC).

To demonstrate the difference between monitoring and hybrid certification models, we first present the monitoring certification model for data integrity-at-rest, expressed by the EC_Assertion monitoring rule R1 that is listed below. The specification of this rule as well as all models in the paper, assumes the following agents and variables denoting them: service consumers (_sc), target of certification (_TOC), authentication infrastructure (_AI), certification authority (_CA).

Rule R1: **Happens**(e(_e1,_sc,_TOC,REQ,_updOp(_cred,_data,_auth),_TOC),t1,[t1,t1]) ^
Happens(e(_e2,_TOC,_AI,RES,_updOp(_cred,_data,_vCode),_TOC),t2,[t1,t1+d1]) ^ (_vCode ≠ Nil) ⇒
Happens(e(_e3,_TOC,_A,REQ,_notifO(_cred,_data,_auth,_h),_TOC),t3,[t2,t2+d2])

According to R1 when a call of an update operation in a _TOC is detected at some time point t1 (see event Happens(e(_e1,_sc,_TOC,REQ,_updOp(_cred,_data,_auth),_TOC),t1,[t1,t1])) and a response to this call occurs after it (see event Happens(e(_e2 ,_TOC, _AI, RES,_updOp(_cred, _data,_verCode),_TOC),t2,[t1,t2+d1])) indicating that the request has been granted (see condition (_vCode ≠ Nil) in the rule), the monitor should also check for the existence of another event showing the call of an operation in some authorisation agent _A to notify the receipt and execution of the update request (see Happens(e(_e3,_TOC,_CA, REQ,_notifO(_cred,_data,_auth,_h),_TOC),t3,[t2,t2+d2]))[14]. The above model has two limitations in providing assurance for the integrity-at-rest property: (1) it cannot capture updates of data that might have been carried out without using the update interface assumed of _TOC (i.e., _updOp(_cred,_data,_vCode)), and (2) it cannot check that the operation _updOp has checked authorisation rights before updating data.

A hybrid model could be used in this case to overcome partially the first of these limitations. More specifically, a hybrid model in this case could be based on periodic testing to detect if stored data have been modified and monitor the periods between the tests that revealed data modifications to check if appropriate notifications have also been sent. Data modifications could be detected by obtaining the hash value of the relevant data file in the TOC periodically. Then, if across the execution of two consecutive tests, the last retrieved hash value of the file is different from the previous hash value, a data modification action can be deduced. In parallel with the execution of this periodic test, the hybrid model will also

[14] Note that the operation signatures used in the rule may change depending on _TOC without affecting the generality of the rule.

monitor the execution of notification operations. Hence, when a data modification action is detected by two consecutive tests, the hybrid model could also check whether a correlated notification operation has been executed within the period between the tests.

This hybrid model model can be expressed using the following monitoring rule and assumption:

Rule R2: **Happens**(e(_e1,_CA,_TOC,EXC(T_{per}), _getHash(_TOC,_file,_h1),_CA), t1, [t1,t1]) ^ **HoldsAt**(LastHash(_file,_h2,t2),t1) ^ (_h1 ≠ _h2) ⇒ **Happens**(e(_e3,_TOC,_CA,REQ,_notifO(_cred,_data, _auth,_h1),_TOC),t3,[t2,t1])

Assumption A1: Happens(e(_e1,_CA,_TOC,REQ, _getHash(_TOC,_file,_h1),_TOC),t1,[t1,t1]) ^ **HoldsAt**(LastHash(_file,_h2,t2),t1) ^ (_h1 ≠ _h2) ⇒

Terminates(_e1,LastHash(_file,_h2,t2),t1) ^ **Initiates**(_e1,LastHash(_file,_h1,t1),t1)

For readability purposes, we also provide the specification of the Rule R2 in the BNF syntax of SecurSLA* below:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
         uuid = url2
         sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS --------------------------------------- */
            party {
             id = in::id::c
             role = in::toc
            }

            abstractparty {
             id = in::id::a
             role = in::certificationauthority
            }

            abstractparty {
             id = in::id::b
             role = in::serviceconsumer
            }

            abstractparty {
             id = in::id::d
             role = id::authenticationinfrastructure
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */

             interfacedecl {
                id = interface::id::c::1
                providerref = in::id::c

                interfacespec {
                    name = in::id::c::tocinterface

                    operation { name = exctestac
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                          input {      name = tper
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
```

```
                           }

       operation { name = rqsac
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }
                           }


       operation { name = rqsbc
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }
                           }
       operation { name = updatehashbc
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }
                           }
       operation { name = gethashac
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }
                           }}
       operation { name = comparehash
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }
                           }}


interfacedecl {
   id = interface::id::a::1
   providerref = in::id::a

   interfacespec {
       name = in::id::ainterface

       operation { name = reqca
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }}
       operation { name = resca
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }}
       operation { name = notifyalterationca
             input {      name = data
                           datatype = url
                           domain = ( equals none)
                           auxiliary = true }}
                           }}

interfacedecl {
   id = interface::id::d::1
   providerref = in::id::d

   interfacespec {
       name = nr::id::dinterface

       operation { name = rspcd
```

```
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}

                    operation { name = exctestad
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    operation { name = authorisead
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    }}

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


exctestacv is ( invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ])

rqsacv is ( invocation [ invoke { endpoint = c222
       operation = rqsac
       param { name = data value = requestop} } ])

rqsbcv is ( invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop} } ])

updatehashbcv is ( invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ])

gethashacv is ( invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = gethashop} } ])

comparehashv is ( invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareopop} } ])

reqcav is ( invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ])

rescav is ( invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ])

notifyalterationcav is ( invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyop} } ])

rspcdv is ( invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ])

exctestadv is ( invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ])

authoriseadv is ( invocation [ invoke { endpoint = d333
       operation = exctestad
       param { name = data value = authoriseop} } ])
```

```
exctestac::series is series (
       invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ] )

rqsac::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsca
       param { name = data value = requestop} } ] )

rqsbc::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop } } ] )

updatehashbc::series is series (
       invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ] )

gethashac::series is series (
       invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = dethashop} } ] )

comparehash::series is series (
       invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareop} } ] )

reqca::series is series (
       invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ] )

resca::series is series (
       invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ] )

notifyalterationca::series is series (
       invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyalterationop} } ] )

rspcd::series is series (
       invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ] )

exctestad::series is series (
       invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ] )

authorisead::series is series (
       invocation [ invoke { endpoint = d333
       operation = authorisead
       param { name = data value = authoriseop} } ] )

/* ---            Guaranteed Terms     ---------- */
agreement_term { id = at::term::1
   precondition { count ( periodic [ exctestac::series ]
                invoke { endpoint = c555
                operation = gethashcv
                param { name = data value = gethashop } }

                invoke { endpoint = c666
                operation = comparehash
                param { name = data value  = notsamehashvalues } } ) }

     guaranteedaction { id = agreement::term::1
                        actor = provider::id::a
                        policy = OPTIONAL
                        trigger = yes
                        invoke { endpoint = a333 operation = notifyalteration } } } } }
```

For readability purposes, we also provide the specification of the Assumption 1 in the BNF syntax of SecurSLA* below:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

     sla_template {
        uuid = url2
        sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS --------------------------------------- */
            party {
             id = in::id::c
             role = in::toc
            }

            abstractparty {
             id = in::id::a
             role = in::certificationauthority
            }

            abstractparty {
             id = in::id::b
             role = in::serviceconsumer
            }

            abstractparty {
             id = in::id::d
             role = id::authenticationinfrastructure
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */

             interfacedecl {
                id = interface::id::c::1
                providerref = in::id::c

                interfacespec {
                    name = in::id::c::tocinterface

                    operation { name = exctestac
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                           input {      name = tper
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }

                    operation { name = rqsac
                           input {      name = data
```

```
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
                            }


          operation { name = rqsbc
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
                            }
          operation { name = updatehashbc
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
                            }
          operation { name = gethashac
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
                            }}
          operation { name = comparehash
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }
                            }}


interfacedecl {
   id = interface::id::a::1
   providerref = in::id::a

   interfacespec {
       name = in::id::ainterface

       operation { name = reqca
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
       operation { name = resca
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
       operation { name = notifyalterationca
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
                            }}

interfacedecl {
   id = interface::id::d::1
   providerref = in::id::d

   interfacespec {
       name = nr::id::dinterface

       operation { name = rspcd
                input {      name = data
                            datatype = url
                            domain = ( equals none)
                            auxiliary = true }}
```

```
                    operation { name = exctestad
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    operation { name = authorisead
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    }}

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


exctestacv is ( invocation [ invoke { endpoint = c111
      operation = exctestac
      param { name = data value = testexecop} } ])

rqsacv is ( invocation [ invoke { endpoint = c222
      operation = rqsac
      param { name = data value = requestop} } ])

rqsbcv is ( invocation [ invoke { endpoint = c333
      operation = rqsbc
      param { name = data value = requestop} } ])

updatehashbcv is ( invocation [ invoke { endpoint = c444
      operation = updatehashbc
      param { name = data value = updateop} } ])

gethashacv is ( invocation [ invoke { endpoint = c555
      operation = gethashac
      param { name = data value = gethashop} } ])

comparehashv is ( invocation [ invoke { endpoint = c666
      operation = comparehash
      param { name = data value = compareopop} } ])

reqcav is ( invocation [ invoke { endpoint = a111
      operation = reqca
      param { name = data value = requestop} } ])

rescav is ( invocation [ invoke { endpoint = a222
      operation = resca
      param { name = data value = responseop} } ])

notifyalterationcav is ( invocation [ invoke { endpoint = a333
      operation = notifyalterationca
      param { name = data value = notifyop} } ])

rspcdv is ( invocation [ invoke { endpoint = d111
      operation = rspcd
      param { name = data value = responseop} } ])

exctestadv is ( invocation [ invoke { endpoint = d222
      operation = exctestad
      param { name = data value = testexecop} } ])

authoriseadv is ( invocation [ invoke { endpoint = d333
      operation = exctestad
      param { name = data value = authoriseop} } ])

exctestac::series is series (
      invocation [ invoke { endpoint = c111
      operation = exctestac
      param { name = data value = testexecop} } ] )
```

```
rqsac::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsca
       param { name = data value = requestop} } ] )

rqsbc::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop } } ] )

updatehashbc::series is series (
       invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ] )

gethashac::series is series (
       invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = dethashop} } ] )

comparehash::series is series (
       invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareop} } ] )

reqca::series is series (
       invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ] )

resca::series is series (
       invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ] )

notifyalterationca::series is series (
       invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyalterationop} } ] )

rspcd::series is series (
       invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ] )

exctestad::series is series (
       invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ] )

authorisead::series is series (
       invocation [ invoke { endpoint = d333
       operation = authorisead
       param { name = data value = authoriseop} } ] )

/* ---            Guaranteed Terms     ---------- */
agreement_term { id = at::term::1
  precondition { count ( difference [ gethashac::series , rqsacv ] greater_than 0 )
  invoke { endpoint = c555
       operation = gethashcv
       param { name = data value = gethashop } }
  invoke { endpoint = c666
       operation = comparehash
       param { name = data value  = notsamehashvalues } } ) }


guaranteedaction { id = agreement::term::1
```

```
                actor = provider::id::a
                policy = OPTIONAL
                trigger = none
                invoke { endpoint = c444 operation = updatehash } } } } }
```

Rule R2 is “hybrid” as it includes normal monitoring events (i.e., REQ and RES events) and events that trigger the execution of tests (i.e., EXC events). R2 expresses a hybrid dependent mode model where evidence arising from testing triggers the acquisition of monitoring evidence. Hence, testing is the primary form of assessment. In particular, R2 forces the execution of the event Happens(e(_e1, _CA, _TOC, EXC(Tper), _getHash(_TOC, _file,_h1),_TOC), t1,[t1,t1]) periodically every Tper time units to invoke the operation _getHash in the testing interface of _TOC and obtain the current hash value (_h1) of the data file (_file) of _TOC. If this value is different from the hash value recorded by a previous test at some t2 (i.e., the value recorded in the fluent LastHash(_file,_h2,t2),t1), rule R2 checks if an update notification has also occurred between t2 and t1, as expressed by the monitoring event Happens(e(_e3,_TOC,_A,REQ,_notifO(_cred,_data,_auth,_h1),_TOC),t3,[t2,t1]). The hybrid model uses also a monitoring assumption (i.e., A1). This assumption is used in the model to update the hash value recorded in the fluent LastHash, if a test retrieves a hash value that is different from the last recorded one.

Although the above model can capture data updates that have taken place without the invocation of the file updating interface, it cannot guarantee that it can capture all possible updates that might have taken place. In particular, it won't be able to detect if more than one updates have taken place between two consecutive executions of the periodic test. Hence, it addresses the first of the limitations of the monitoring problem (i.e., limitation (1)) only partially.

To address the second limitation of the monitoring model (i.e., limitation (2)), it is possible to construct a different hybrid model. This model could rely on testing to ensure that every time that an agent that requests a data alteration, it has the authorisation right to do the requested alteration. This model can be expressed by the monitoring rule below:

Rule R3: Happens(e(_e1,_sc,_TOC,REQ,_updOp(_cred,_data, _auth),_TOC),t1,[t1,t1]) ^
Happens(e(_e2,_TOC,_AI,RES,_updOp(_cred,_data,_vCode1),_TOC),t2,[t1,t1+d1]) ^ (_vCode1 ≠ Nil) ⇒
Happens(e(_e3,_CA,_AI,EXC,_authorO(_cred,_auth,_vCode2),_TOC),t3,[t2,t2+d2])^(_vCode2≠Nil)

For readability purposes, we also provide the specification of the Rule R3 in the BNF syntax of SecurSLA* below:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
        uuid = url2
        sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS -------------------------------------- */
            party {
             id = in::id::c
             role = in::toc
            }

            abstractparty {
             id = in::id::a
             role = in::certificationauthority
            }

            abstractparty {
             id = in::id::b
             role = in::serviceconsumer
            }

            abstractparty {
             id = in::id::d
             role = id::authenticationinfrastructure
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */

             interfacedecl {
                id = interface::id::c::1
                providerref = in::id::c

                interfacespec {
                    name = in::id::c::tocinterface

                    operation { name = exctestac
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                           input {      name = tper
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }

                    operation { name = rqsac
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }


                    operation { name = rqsbc
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }
                    operation { name = updatehashbc
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }
                    operation { name = gethashac
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }}
                    operation { name = comparehash
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }
                                         }}


             interfacedecl {
                id = interface::id::a::1
                providerref = in::id::a

                interfacespec {
                    name = in::id::ainterface
                    operation { name = reqca
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }}
                    operation { name = resca
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }}
                    operation { name = notifyalterationca
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }}
                                         }}

             interfacedecl {
                id = interface::id::d::1
                providerref = in::id::d

                interfacespec {
                    name = nr::id::dinterface

                    operation { name = rspcd
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true }}

                    operation { name = exctestad
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true } }
                    operation { name = authorisead
                           input {      name = data
                                         datatype = url
                                         domain = ( equals none)
                                         auxiliary = true } }
                    }}

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


exctestacv is ( invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ])

rqsacv is ( invocation [ invoke { endpoint = c222
       operation = rqsac
       param { name = data value = requestop} } ])

rqsbcv is ( invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop} } ])

updatehashbcv is ( invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ])

gethashacv is ( invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = gethashop} } ])

comparehashv is ( invocation [ invoke { endpoint = c666
       operation = comparehash
```

```
        param { name = data value = compareopop} } ])

reqcav is ( invocation [ invoke { endpoint = a111
        operation = reqca
        param { name = data value = requestop} } ])

rescav is ( invocation [ invoke { endpoint = a222
        operation = resca
        param { name = data value = responseop} } ])

notifyalterationcav is ( invocation [ invoke { endpoint = a333
        operation = notifyalterationca
        param { name = data value = notifyop} } ])

rspcdv is ( invocation [ invoke { endpoint = d111
        operation = rspcd
        param { name = data value = responseop} } ])

exctestadv is ( invocation [ invoke { endpoint = d222
        operation = exctestad
        param { name = data value = testexecop} } ])

authoriseadv is ( invocation [ invoke { endpoint = d333
        operation = exctestad
        param { name = data value = authoriseop} } ])

exctestac::series is series (
        invocation [ invoke { endpoint = c111
        operation = exctestac
        param { name = data value = testexecop} } ] )

rqsac::series is series (
        invocation [ invoke { endpoint = c222
        operation = rqsca
        param { name = data value = requestop} } ] )

rqsbc::series is series (
        invocation [ invoke { endpoint = c333
        operation = rqsbc
        param { name = data value = requestop } } ] )

updatehashbc::series is series (
        invocation [ invoke { endpoint = c444
        operation = updatehashbc
        param { name = data value = updateop} } ] )

gethashac::series is series (
        invocation [ invoke { endpoint = c555
        operation = gethashac
        param { name = data value = dethashop} } ] )

comparehash::series is series (
        invocation [ invoke { endpoint = c666
        operation = comparehash
        param { name = data value = compareop} } ] )

reqca::series is series (
        invocation [ invoke { endpoint = a111
        operation = reqca
        param { name = data value = requestop} } ] )

resca::series is series (
        invocation [ invoke { endpoint = a222
        operation = resca
        param { name = data value = responseop} } ] )

notifyalterationca::series is series (
        invocation [ invoke { endpoint = a333
```

```
       operation = notifyalterationca
       param { name = data value = notifyalterationop} } ] )

rspcd::series is series (
       invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ] )

exctestad::series is series (
       invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ] )

authorisead::series is series (
       invocation [ invoke { endpoint = d333
       operation = authorisead
       param { name = data value = authoriseop} } ] )


/* ---            Guaranteed Terms     ---------- */
agreement_term { id = at::term::1
   precondition { count ( difference [ rqsbc::series , rspcd ] greater_than 0 )
   invoke { endpoint = d111
        operation = updatehashbc
        param { name = data value = updateop } }
) }


guaranteedaction { id = agreement::term::1
                actor = provider::id::a
                policy = OPTIONAL
                trigger = reply [invoke { endpoint = d222
                                operation = exctestad
                                param { name = data value = testexecop]
                invoke { endpoint = d333 operation = authorisead } }
 } } }
```

Rule R3 monitors requests for updates of _TOC data through its normal updating interface. However, for every such request that is granted by _TOC, it requests the execution of a test to check if the entity that requested the update had indeed the authorisation to update data. This is expressed by the EXC event Happens(e(_e3,_CA,_AI,EXC,_authorO(_cred,_auth,_verCode2),_TOC),t3,[t2,t2+d2])) and the condition _verCode2 ≠ Nil. In R3, the monitoring evidence triggers the execution of tests. Hence, the rule expresses a dependent hybrid model where monitoring is the primary form of assessment. Rules R2 and R3 are examples of general time correlation structures that may arise in dependent hybrid certification model and which are shown in Figure 62.

Part (a) of the figure shows dependent hybrid certification models where testing is the dominant form of assessment. In such models, test plans each consisting of a series of tests (i.e., $\{Test_{n1},…,Test_{nL}\}$) are executed according to some periodic schedule. Assuming that the execution of a test plan starts at $t_s^{(n)}$ and ends at $t_e^{(n)}$, the hybrid model may also check for monitoring events that occurred within the interval $[t_s^{(n)}–d_1, t_e^{(n)}+d_2]$ in order to provide an assessment of the security property of interest. Note that the length of the execution of each test plan and the monitoring events found within $[t_s^{(n)}–d_1, t_e^{(n)}+d_2]$ may vary.

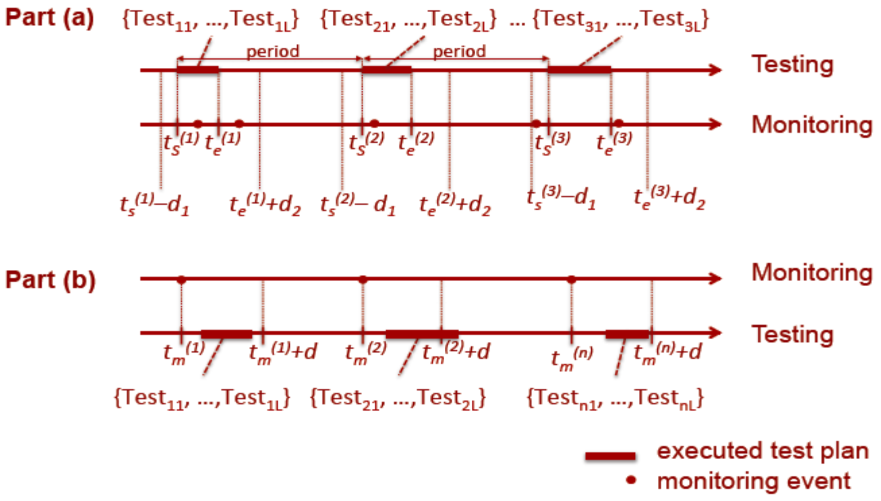


FIGURE 62 – DEPENDENT MODE HYBRID CERTIFICATION MODELS

Part (b) of the figure shows the timelines of evidence collection in dependent hybrid certification models where monitoring is the dominant form of assessment. In such models following the collection of monitoring evidence (events), tests plans are executed to cross-check/complete it. The execution of these plans starts within the range [$t_m^{(n)}$, $t_m^{(n)}$+d] where $t_m^{(n)}$ is the time of occurrence of the last event in a pattern of events that should trigger the execution of the plan and d is a period set by the model. The length of the execution of each test plan may vary.

Example 2: Hybrid, independent mode models

Our second example shows the use of a hybrid approach in certifying cloud service availability. As defined in (CSA, D2.1 Development of security properties specification scheme and security dependency models, 2013) this property expresses the ability of a TOC to produce a non-faulty response within a certain period of time and is measured by the percentage of calls that satisfy this condition over an assessment period. An independent hybrid model for the certification of TOC availability could be based on collecting evidence regarding the availability of a TOC through monitoring and testing independently (i.e., without any of these activities being triggered by outcomes of the other) and then correlating and cross-checking the collected pools of evidence to produce a hybrid assessment of the property. More specifically, the hybrid model could include monitoring formulas to record instances of invocation of TOC operators where TOC produced a response within the acceptable time limit and the instances where it did not, and keep a record of counters of these instances from which an overall availability measure could be drawn. The formulas that could be used to collect this monitoring evidence are as follows:

Assumption A2 (monitoring evidence):
Happens(e(_e1,_CA,_TOC,REQ,_OP(_data),_TOC),t1,[t1,t1])^ **Happens**(e(_e2,_TOC,_CA,RES,_OP(_data),_TOC), t2,[t1,t1+t_{av}]) ^
HoldsAt(MCounterA(_TOC,_MCA),t2)⇒
Terminates(_e1,MCounterA(_TOC, _MCA), t2) ^
Initiates(_e1,MCounterA(_TOC, _MCA+1), t2)^
Initiates(_e1,MAvail(_TOC,_OP(_data),t2–t1), t2)

Assumption A3 (monitoring evidence):
Happens(e(_e1,_CA,_OC,REQ,_OP(_data),_TOC),t1,[t1,t1]) ^¬ **Happens**(e(_e2,_TOC,_CA,RES,_OP(_data),_TOC),t2,[t1,t1+t_{av}]))^
HoldsAt(MCounterU(_TOC,_MCU), t2) ⇒
Terminates(_e1,MCounterU(_TOC,_MCU), t2) ^
Initiates(_e1,MCounterU(_TOC, _MCU+1), t2) ^
Initiates(_e1,MUnav(_TOC,_OP(_data),t2–t1),t2)

For readability purposes, we also provide the specification of the Assumption A2 in the BNF syntax of SecurSLA* below
:

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

     sla_template {
        uuid = url2
        sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS --------------------------------------- */
            party {
             id = in::id::c
             role = in::toc
            }

            abstractparty {
             id = in::id::a
             role = in::certificationauthority
            }

            abstractparty {
             id = in::id::b
             role = in::serviceconsumer
            }

            abstractparty {
             id = in::id::d
             role = id::authenticationinfrastructure
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */

             interfacedecl {
                id = interface::id::c::1
                providerref = in::id::c

                interfacespec {
                    name = in::id::c::tocinterface

                    operation { name = exctestac
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                           input {      name = tper
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }

                    operation { name = rqsac
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }


                    operation { name = rqsbc
                           input {      name = data
                                        datatype = url
```

```
                        domain = ( equals none)
                        auxiliary = true }
                        }
        operation { name = updatehashbc
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }
                        }
        operation { name = gethashac
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }
                        }}
        operation { name = comparehash
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }
                        }}


interfacedecl {
   id = interface::id::a::1
   providerref = in::id::a

   interfacespec {
       name = in::id::ainterface

        operation { name = reqca
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }}
        operation { name = resca
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }}
        operation { name = notifyalterationca
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }}
                        }}

interfacedecl {
   id = interface::id::d::1
   providerref = in::id::d

   interfacespec {
       name = nr::id::dinterface

        operation { name = rspcd
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true }}

        operation { name = exctestad
              input {      name = data
                        datatype = url
                        domain = ( equals none)
                        auxiliary = true } }
        operation { name = authorisead
              input {      name = data
                        datatype = url
```

```
                                        domain = ( equals none)
                                        auxiliary = true } }
                    }}

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


exctestacv is ( invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ])

rqsacv is ( invocation [ invoke { endpoint = c222
       operation = rqsac
       param { name = data value = requestop} } ])

rqsbcv is ( invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop} } ])

updatehashbcv is ( invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ])

gethashacv is ( invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = gethashop} } ])

comparehashv is ( invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareopop} } ])


reqcav is ( invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ])

rescav is ( invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ])

notifyalterationcav is ( invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyop} } ])

rspcdv is ( invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ])

exctestadv is ( invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ])

authoriseadv is ( invocation [ invoke { endpoint = d333
       operation = exctestad
       param { name = data value = authoriseop} } ])

exctestac::series is series (
       invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ] )

rqsac::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsca
       param { name = data value = requestop} } ] )

rqsbc::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop } } ] )

updatehashbc::series is series (
       invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ] )

gethashac::series is series (
       invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = dethashop} } ] )

comparehash::series is series (
       invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareop} } ] )

reqca::series is series (
       invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ] )

resca::series is series (
       invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ] )

notifyalterationca::series is series (
       invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyalterationop} } ] )

rspcd::series is series (
       invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ] )

exctestad::series is series (
       invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ] )

authorisead::series is series (
       invocation [ invoke { endpoint = d333
       operation = authorisead
       param { name = data value = authoriseop} } ] )

rsqac::counts is list[ index = request , type(integer) ]
rsqac::avail is list[ index = request , type(integer) ]



/* ---            Guaranteed Terms     ---------- */

agreement_term { id = at::term::2
       precondition {
       count ( difference [ rqsac::series , rqsacv::avail ] greater_than t )
       }
       guaranteedstate { id = gstate1
       update(rsqac_counts, rsqac::counts::call equals rqsacv::data ,
              add ( rsqac::counts, 1))
       update(rsqac_avail, rsqac::counts::call equals rqsacv::data ,
              add ( rsqac::avail, 1))


           }
       }
```

```
    }
}
```

The first of the above monitoring formulas (i.e., assumption A2) monitors calls to any operation in a _TOC and the responses to them (see events Happens(e(_e1, _CA, _TOC, REQ, _OP(_data),_TOC), t1, R(t1,t1)) and Happens(e(_e2, _CA, _TOC, RES, _OP(_data), _TOC), t2,[t1,t1+t_{av}])) and if a response is within the required period (t_{av}), it updates the counter of instances where _TOC was available and records the related call (in fluents MCounterA(_TOC,_MCA,t2) and MAvail(_TOC,_OP(_data),t2–t1), respectively). The second formula (i.e., assumption A3) monitors calls to _TOC operations that did not produce a response within the required time, and keeps an overall counter of unavailability and the related calls in fluents MCounterU(_TOC,_MCU,t2) and MUnav(_TOC,_OP(_data),t2–t1)).

The hybrid model for the certification of availability could also incorporate a test-based availability assessment sub-model. This sub-model can execute a randomly selected operation in the interface of _TOC periodically to check its availability, and keep a record of instances of test-triggered invocations of operations of TOC in which a response was produced within the required time period, and instances of test-triggered invocations where it was not.
This sub-model is expressed by following formulas for collecting testing evidence:

<u>Assumption A4 (testing evidence):</u>
Happens(e(_e1,_CA,_TOC, EXC(T_{per}), _x=random(interface(_TOC)),_TOC),t1,[t1,t1]) ^
Happens(e(_e2,_TOC, _CA RES,_x,_TOC),t2,[t1,t1+t_{av}])^
HoldsAt(TCounterA(_TOC,_TCA),t2)
⇒ **Terminates**(_e1,TCounterA(_TOC,_TCA),t2) ^
Initiates(_e1,TCounterA(_TOC,_TCA+1),t2)^
Initiates(_e1,TAvail(_TOC,_x,t2–t1),t2)

<u>Assumption A5 (testing evidence):</u>
Happens(e(_e1,_CA,_TOC, EXC(T_{per}), _x=random(interface(_TOC)),_TOC),t1,[t1,t1]) ^
¬ **Happens**(e(_e2,, _TOC, _CA, RES,_x,_TOC),t2, [t1,t1+t_{av}])^ **HoldsAt**(TCounterU(_TOC,_TCU),t2) ⇒
Terminates(_e1,TCounterU(_TOC,_TCU),t2) ^
Initiates(_e1,TCounterU(_TOC,_TCU+1),t2) ^
Initiates(_e1,TUnav(_TOC,_x, t2–t1),t2)

For readability purposes, we also provide the specification of the Assumption A4 in the BNF syntax of SecurSLA* below

```
assertion {
   agreedAt = n/a
   effectiveFrom = 00 : 00 : 00 : 00
   effectiveUntil =  23 : 59 : 59 : 59
   templateId = url1

      sla_template {
         uuid = url2
         sla_model_version = sla_at_soi_sla_model_v1.0


/* ---- PARTY DESCRIPTIONS --------------------------------------- */
             party {
               id = in::id::c
               role = in::toc
             }

             abstractparty {
               id = in::id::a
               role = in::certificationauthority
             }

             abstractparty {
               id = in::id::b
               role = in::serviceconsumer
             }

             abstractparty {
               id = in::id::d
             role = id::authenticationinfrastructure
            }

/* ---- TOC INTERFACE DECLARATIONS ------------------- */

             interfacedecl {
                id = interface::id::c::1
                providerref = in::id::c

                interfacespec {
                    name = in::id::c::tocinterface

                    operation { name = exctestac
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                           input {      name = tper
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }

                    operation { name = rqsac
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }


                    operation { name = rqsbc
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }
                    operation { name = updatehashbc
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }
                    operation { name = gethashac
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }}
                    operation { name = comparehash
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }
                                        }}


             interfacedecl {
                id = interface::id::a::1
                providerref = in::id::a

                interfacespec {
                    name = in::id::ainterface

                    operation { name = reqca
                           input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}
                    operation { name = resca
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}
                    operation { name = notifyalterationca
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}
                                        }}

             interfacedecl {
                id = interface::id::d::1
                providerref = in::id::d

                interfacespec {
                    name = nr::id::dinterface

                    operation { name = rspcd
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true }}

                    operation { name = exctestad
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    operation { name = authorisead
                          input {      name = data
                                        datatype = url
                                        domain = ( equals none)
                                        auxiliary = true } }
                    }}

/* ---- VARIABLE DECLARATIONS----------------------------------------- */


exctestacv is ( invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ])

rqsacv is ( invocation [ invoke { endpoint = c222
       operation = rqsac
       param { name = data value = requestop} } ])

rqsbcv is ( invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop} } ])

updatehashbcv is ( invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ])

gethashacv is ( invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = gethashop} } ])

comparehashv is ( invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareopop} } ])

reqcav is ( invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ])
```

```
rescav is ( invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ])

notifyalterationcav is ( invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyop} } ])

rspcdv is ( invocation [ invoke { endpoint = d111
       operation = rspcd
       param { name = data value = responseop} } ])

exctestadv is ( invocation [ invoke { endpoint = d222
       operation = exctestad
       param { name = data value = testexecop} } ])

authoriseadv is ( invocation [ invoke { endpoint = d333
       operation = exctestad
       param { name = data value = authoriseop} } ])

exctestac::series is series (
       invocation [ invoke { endpoint = c111
       operation = exctestac
       param { name = data value = testexecop} } ] )

rqsac::series is series (
       invocation [ invoke { endpoint = c222
       operation = rqsca
       param { name = data value = requestop} } ] )

rqsbc::series is series (
       invocation [ invoke { endpoint = c333
       operation = rqsbc
       param { name = data value = requestop } } ] )

updatehashbc::series is series (
       invocation [ invoke { endpoint = c444
       operation = updatehashbc
       param { name = data value = updateop} } ] )

gethashac::series is series (
       invocation [ invoke { endpoint = c555
       operation = gethashac
       param { name = data value = dethashop} } ] )

comparehash::series is series (
       invocation [ invoke { endpoint = c666
       operation = comparehash
       param { name = data value = compareop} } ] )

reqca::series is series (
       invocation [ invoke { endpoint = a111
       operation = reqca
       param { name = data value = requestop} } ] )

resca::series is series (
       invocation [ invoke { endpoint = a222
       operation = resca
       param { name = data value = responseop} } ] )

notifyalterationca::series is series (
       invocation [ invoke { endpoint = a333
       operation = notifyalterationca
       param { name = data value = notifyalterationop} } ] )

rspcd::series is series (
       invocation [ invoke { endpoint = d111
```

```
        operation = rspcd
        param { name = data value = responseop} } ] )

exctestad::series is series (
        invocation [ invoke { endpoint = d222
        operation = exctestad
        param { name = data value = testexecop} } ] )

authorisead::series is series (
        invocation [ invoke { endpoint = d333
        operation = authorisead
        param { name = data value = authoriseop} } ] )

rsqac_record is list[index = response , invocation [ invoke { endpoint = a111 operation =
reqcav param { name = data value = testexec} } ] , timeof [invocation [ invoke { endpoint
= d222 operation = exvtestad param { name = data value = testexecop} } ] ] ]
rescac::avail is list[ index = request , type(integer) ]

/* ---           Guaranteed Terms     ---------- */

agreement_term { id = at::term::3

        precondition {
        count ( difference [ rqsac::series , rqsacv::avail ] greater_than t )
        }
        guaranteedstate { id = gstate3

        insert(rsqac_record, rsqac::record::id equals rsqacv::data ,
                rsqacv, timeof [ rqstcv ])
        update(rescac::avail, rescac::counts::call equals rs rescac qacv::data ,
                add (rescac::avail, 1))


                 }

              }
          }
}
```

A4 and A5 are similar to assumptions A2 and A3 respectively except that, instead of monitoring real operation calls, they execute a randomly selected operation in the interface of _TOC periodically (see the event Happens(e(_e1,_CA,_TOC,EXC(T_{per}),_x=random(interface(_TOC)), _TOC),t1,[t1,t1])) to check its availability, and update fluents recording the overall counters of availability and unavailability of _TOC and the test executions that revealed them.

In the hybrid model, the assumption pairs (A2, A3), and (A4, A5) are used to collect evidence independently without any monitoring events triggering tests or vice versa. However, it might still be desirable to correlate the testing and monitoring evidence. For example:

a) The overall availability measure may be computed on the basis of both test and monitoring evidence as A = (_MCA + _TCA)/(_MCA+ _TCA+_MCU+ _TCU).

b) Testing events may be considered as valid evidence of TOC availability only if all real calls of TOC in the range [tmon–d, tmon+d] produced responses within tav.

c) Monitored calls to TOC that produced a response within the maximum allowed period tav may be disregarded in computing TOC's availability if the relevant responses were marginally below tav and all testing events for the same TOC in the range [tmon–d,tmon+d] (tmon is the timestamp of a monitored call) produced responses after the maximum allowed period tav.

d) An availability measure based on testing (monitoring) evidence will be used for issuing a certificate only if the availability measure based on monitoring (testing) evidence over the same period is no more than 1% different from it.

Clearly, several other combinations of monitoring and testing evidence may be defined. In general, in an independent hybrid certification model:

- individual (or groups of) instances of monitoring and testing evidence may be cross-validated against each other before producing an overall assessment on the basis of any of these types of evidence (see (b) and (c) above);
- aggregate assessments based on each type of evidence may be validated against an aggregate assessment based on the other type before issuing a certificate (see (d) above);
- or an aggregate assessment may be formulated from both monitoring and testing evidence as in case (a) above.

Compared to non-hybrid models for certifying availability, the hybrid model introduced above can produce availability assessments of higher confidence as the monitoring and testing evidence can be cross-checked before being used in an assessment (and certificate) and can both be included in a certificate depending on the chosen validation checks. Hybrid models offer also a more extended pool of evidence and possibilities to decide which data are relevant and of sufficient quality so that they can be taken into consideration for issuing a hybrid certificate. Apart from increasing the confidence level of assessments, hybrid models are also more customisable than traditional certification models since they offer the choice of deciding how test and monitoring evidence should be correlated, cross-checked and used in assessments.

7.3.1. Comparison between hybrid and traditional certification models

This section has no corresponding section in D2.2. It contains a comparison between Hybrid Certification Models and the traditional Certification Models used in the industry.

In this section we present a comparison between hybrid and traditional certification of cloud services. Traditional approaches for certifying security properties rely on manual inspections and audits, proving to be static, inflexible, non-automated and unable to realise the economic dimension that the Cloud entails (Windhorst I., 2013). As already stated, CSA has generated the STAR self-assessment certification framework that allows cloud providers to submit self-assessment reports, when fully implemented (J. Reavis, 2012). At the same time, CSA has generated the Cloud Controls Matrix (CCM) (CSA, https://cloudsecurityalliance.org/research/ccm/), which provides a framework of controls that gives detailed understanding of security concepts and principles that are aligned to the Cloud Security Alliance guidance in 13 domains (CSA, https://cloudsecurityalliance.org/star/self-assessment/). CCM facilitates regulatory compliance and provides organizations with the needed structure, detail and clarity relating to information security tailored to the cloud industry. It is also specifically designed to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider, integrating the ISO/IEC 27001 management systems standard (CSA, https://cloudsecurityalliance.org/research/ccm/). However, CCM is human-process centric requiring from the companies that adopt it to address the issues that they define critical concerning cloud security and to pre-assess how mature their systems are (CSA, https://cloudsecurityalliance.org/research/ccm/) (J. Reavis, 2012). CCM enables the integration, monitoring and managing of cloud services through a framework that can take care of the elementary issues regarding cloud security (CSA, https://cloudsecurityalliance.org/research/ccm/), but it does not support certification as an automated service in the cloud (Saxena, 2013).

Traditional certification models (i.e. ISO/IEC 27001, NIST) also require manual inspections and are unable to provide the required level of assurance in cloud computing and to fit the dynamic nature of the cloud, focusing on monolithic software components (S. Cimato, 2013) and failing to address on-demand self-service, dynamic allocation of resources and multi-tenancy (Windhorst I., 2013) (Kaliski B. Jr., 2010). Additionally, traditional certification models lack in trust, transparency and accuracy, as they do not support the constant provision of information about the security of cloud services, unlike our hybrid approach that relies on incremental monitoring and automated testing and it is focused on cloud services.

IT audits have been widely used in providing security assurance. Security auditing approaches focus on verifying the implementation of information security management process and then, evidence is collected and evaluated. According to (Z. Chen, 2010) auditing approaches focus on verifying the application of controls and best practices and are not automated. One more drawback is that they require that the consumer relies on third-party auditors for security assurance. Common Criteria (CC) certification uses Evaluation Technical Reports (Kaluvuri S.P., 2013). CC has also a human-centric approach, unlike our model, which is not designed to support automated security certification, targeting static, monolithic systems and requiring a large investment of resources (S. Cimato, 2013).

8. Conclusion

During the first year of CUMULUS project, the work in Work Package 2 has been focused on the specifications of the three basic certificate categories for the definition of Test-based, Monitoring-based, and TC-based certification models. To better integrate the different models of certification, soon emerged the need of a common vocabulary of terms: the meta-model, proposed for version 1 of this deliverable, tried to unify the different views and give a shared representation of the domain where all the conceptual entities involved in the certification process can be defined. Also a new “TC Support” model has been developed, which is not used as an independent model but is rather providing the trust that is needed for the validation of the Monitoring and Test Based certification

During the second year of CUMULUS project, the three basic Certification Models have been further refined to take into account some architectural changes and the security property vocabulary developed in Work Package 2 (CSA, 2013) was fully integrated in the models.

9. References

(2012). *Common Criteria v3.1 R4 Part 1.*

Anisetti, A. &. (2011). Fine-grained modeling of web services for test-based security certification. *Prec of SCC 2011, Washington DC, USA*.

Anisetti, M., Ardagna, C., & Damiani, E. (2011). Fine-grained modeling of web services for test-based security certification. *In Proc. of the 8th International Conference on Service Computing (SCC 2011)*.

Anisetti, M., Ardagna, C., & Damiani, E. (2011). *Fine-grained modeling of web services for test-based security certification.* Washington, DC, USA: Proc. of SCC.

Chung, L., & Leite, J. (2009). Conceptual modeling. *Foundations and applications.*, 363–379.

Chung, L., Nixon, B., Yu, E., & Mylopoulos, J. (2000). Non-Functional Requirements in Software Engineering,. *5*.

CITY University. (2013). *D3.1 Core Certification Mechanisms.* CUMULUS project.

Consortium, C. (2013). *Security-aware SLA specificaton language and cloud security dependency model - Deliverable 2.1.*

CSA. (n.d.). Retrieved from https://cloudsecurityalliance.org/star/self-assessment/.

CSA. (n.d.). Retrieved from https://cloudsecurityalliance.org/research/ccm/.

CSA. (2013). *D2.1 Development of security properties specification scheme and security dependency models.* CUMULUS project.

CSA. (n.d.). *https://cloudsecurityalliance.org/star/self-assessment/.*

CUMULUS Consortium. (2012). *DoW.*

CUMULUS Consortium. (2015). *D2.4 Final Certification Models.*

Damiani, E., Anisetti, M., & Ardagna, C. (2011). Design and description of evidence-based certificates artifacts for services. *Assert4SOA*.

Damiani, E., Ardagna, C., & Bezzi, M. (2012). Cluster Workshop on "Security Contracts". *CSP EU Forum 2012*.

Damiani, E., Ardagna, C., & Ioini, N. (2009). Open source systems security certification.

E. Damiani, C. A. (2008). *Open Source Systems Security Certification.* Springer.

Feng, J. C. (2011). A Fair Multi-Party Non-Repudiation Scheme for Storage Clouds. *In Proceedings of the International Conference on Collaboration Technologies and Systems (CTS)*, 457-465.

Gurgens, S. R. (2005). On the Security of Fair Non-Repudiation Protocols. *International Journal of Information Security*, 253-262.

Irvine, C., & Levin, T. (1991). Toward a taxonomy and costing method for security services. *Proc. of the 15th Annual Conference on Computer Security Applications (ACSAC 1999).*

J. Reavis, D. C. (2012, Aug). Open Certification Framework. Vision Statement, Cloud Security Alliance.

Kaliski B. Jr., P. W. (2010). Toward Risk Assessment as a Service in Cloud Environments. *Proceeding HotCloud 10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*.

Kaluvuri S.P., K. H. (2013). Security Assurance of Services through Digital Security Certificates. *Proc. of the 20th IEEE International Conference on Web Services (ICWS 2013)*.

M. Anisetti, C. A. (2012). A Low-Cost Security Certification Scheme for Evolving Services. *in Proc. of the 19th IEEE International Conference on Web Services (ICWS 2012), Honolulu, HI, USA*.

M. Anisetti, C. A. (2013). A Test-based Security Certification Scheme for Web Services. *ACM Transactions on the Web*.

Ma., W. W. (2013). A TPA Based Efficient Non-Repudiation Scheme for Cloud Storage. *In Proceedings of the 2nd International Conference On Systems Engineering and Modeling (ICSEM -13)*, 1630-1635.

Markowitch, O. a. (1999). Probabilistic Non-Repudiation without trusted third party. *In Second Conference on Security in Communication Networks '99, Italy*.

Reavis J., C. D. (2012, Aug). "Cloud Security Alliance - Open Certification Framework Vision Statement".

S. Cimato, E. D. (2013). Towards the Certification of Cloud Services. *IEEE 9th World Congress on Services*.

Saxena, S. (2013). Ensuring Cloud Security Using Cloud Control Matrix.

TrustedComputingGroup. (2011). *TPM 1.2 Protection Profile*. Retrieved 09 25, 2013, from Trusted Computing Group: https://www.trustedcomputinggroup.org/

University of Milan. (2014). *D3.2 Core Certification Mechanisms.*

VV. AA. (2011). *TCG Attestation PTS Protocol: Binding to TNC IF-M Specification Version 1.0 Revision 28;.* Retrieved from www.trustedcomputinggroup.org

Windhorst I., S. A. (2013). Dynamic Certification of Cloud Services. *Eighth International Conference on Availability, Reliability and Security (ARES).*

Z. Chen, J. Y. (2010). IT Auditing to assure a secure cloud computing. *IEEE 6th World Congress on Services*.

Zhou, J. a. (1996). A Fair Non-Repudiation Protocol. *In Proceedings of IEEE Symposium on Security and Provacy, pp. 55-61, California*.

Zhou, J. a. (1997). An Efficient Non-Repudiation Protocol. *In Proceedings of the 10th Computer Security Foundations Workshop, IEEE Computer, Okland, USA*, 126-132.

10. Appendix

10.1. Test-based Certification Model generic schema (.xsd file)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" vc:maxVersion="1.1" vc:minVersion="1.0"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
  <xs:annotation>
    <xs:documentation>Created with EditiX (http://www.editix.com) at Thu Feb 24 10:21:29 CET 2011</xs:documentation>
  </xs:annotation>
  <xs:element name="TestBasedCertificationModel" type="testBasedCertificationModelType"> </xs:element>
  <xs:complexType name="testBasedCertificationModelType">
    <xs:sequence maxOccurs="1">
      <xs:element name="CertificationModelId" type="certificationModelType" minOccurs="1"/>
      <xs:element name="Collectors" type="collectorType"> </xs:element>
      <xs:element name="LifeCycle" type="lifeCycleType"/>
      <xs:element name="Toc" type="tocType" maxOccurs="1"/>
      <xs:element name="SecurityProperty" type="securityPropertyType"/>
      <xs:element name="Signature" type="testerType"/>
      <xs:element name="Context" type="contextType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="certificationModelType">
    <xs:sequence>
      <xs:element name="CmId" type="xs:ID"/>
    </xs:sequence>
  </xs:complexType>
  <xs:attribute name="AggregatorDescriptor">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="CHAINING"/>
        <xs:enumeration value="MOST_RECENT"/>
        <xs:enumeration value="MOST_CRITICAL"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:complexType name="agentConfigurationType">
    <xs:sequence maxOccurs="1" minOccurs="0">
      <xs:element name="AgentURI" type="xs:anyURI"/>
      <xs:element name="AgentDesription" type="xs:string" minOccurs="0"/>
      <xs:element name="AgentIntegrityProtection" type="TC_IntegrityProtectionType"/>
    </xs:sequence>
    <xs:attribute name="AgentIsPresent" type="xs:boolean" use="required"/>
    <xs:attribute name="Id" type="xs:ID" use="required"/>
  </xs:complexType>
  <xs:complexType name="actionabilityType">
    <xs:sequence>
      <xs:element name="Description" type="xs:string"> </xs:element>
      <xs:element name="Mechanism" type="xs:string"/>
      <xs:element name="MechanismURI" type="xs:anyURI"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="aggregatorType">
    <xs:sequence>
      <xs:element name="ModelLink" type="xs:anyURI"/>
      <xs:element name="TestMetric" type="testMetricsType"/>
      <xs:element name="ElementForExtension" type="elementForExtensionType"/>
    </xs:sequence>
    <xs:attribute name="AggregatorDescription" type="xs:string"> </xs:attribute>
  </xs:complexType>
  <xs:complexType name="certificateInfoType">
    <xs:sequence maxOccurs="1" minOccurs="1">
      <xs:element name="CertificateType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Test-based"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CertificationDate" type="xs:dateTime"/>
      <xs:element name="OnlineOfflineMode">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Offline/Static"/>
            <xs:enumeration value="Online/Dynamic"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="contextType">
   <xs:attribute name="ConfigurationDate" type="xs:date"/>
</xs:complexType>
<xs:complexType name="collectorType">
   <xs:sequence>
      <xs:element name="AbstractCollector" type="abstracCollectorType" maxOccurs="unbounded"
         minOccurs="1"/>
      <xs:element name="Collector" type="GeneralCollectorType" minOccurs="0"
         maxOccurs="unbounded"> </xs:element>
      <xs:element name="EventBusCollector" type="eventBusCollectorType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="capabilityType">
   <xs:sequence maxOccurs="unbounded">
      <xs:element name="Attacker" type="xs:string"> </xs:element>
      <xs:element name="AttackName" type="xs:string"/>
      <xs:element name="AttackerCapabilities">
         <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
               <xs:element name="CapabilityId" type="xs:integer"/>
               <xs:element name="AttackerCapability" type="xs:string"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="conditionForSomministrationType">
   <xs:choice>
      <xs:element name="ThresholdTraffic" type="thresholdTrafficType"/>
      <xs:element name="Event" type="eventType"/>
      <xs:element name="OtherCondition" type="xs:string"/>
      <xs:element name="DeltaTime" type="xs:time"/>
   </xs:choice>
</xs:complexType>
<xs:complexType name="GeneralCollectorType">
   <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="AgentURI" type="xs:anyURI"/>
      <xs:element name="ConditionForSomministration" type="conditionForSomministrationType"
         maxOccurs="unbounded" minOccurs="0"> </xs:element>
      <xs:element minOccurs="1" name="AbstractCollector" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="ExpirationTime" type="xs:gYearMonth" use="required"/>
   <xs:attribute name="Descriptor" type="xs:string"/>
   <xs:attribute name="Id" type="xs:ID"/>
   <xs:attribute name="isStatic" type="xs:boolean"/>
   <xs:attribute name="toDeploy" type="xs:boolean"/>
</xs:complexType>
<xs:complexType name="eventBusCollectorType">
   <xs:sequence maxOccurs="unbounded">
      <xs:element name="EventId" type="xs:ID"/>
      <xs:element name="EventCode" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="eventType">
   <xs:sequence>
      <xs:element name="Action" type="xs:string"> </xs:element>
      <xs:element name="Condition" type="xs:string"/>
      <xs:element name="Value" type="xs:float"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="elementForExtensionType">
   <xs:sequence>
      <xs:element name="Environment"> </xs:element>
      <xs:element name="TestingTool"/>
      <!-- <xs:element name="KeyInfo" minOccurs="0"/> -->
      <xs:element name="Code"/>
      <xs:element name="Others"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="KeyInfoType" mixed="true">
   <xs:choice maxOccurs="unbounded">
      <xs:element name="KeyName"/>
      <xs:element name="KeyValue"/>
      <xs:element name="RetrievalMethod"/>
      <xs:element name="X509Data"/>
      <xs:element name="PGPData"/>
      <xs:element name="SPKIData"/>
```

```
         <xs:element name="MgmtData"/>
         <xs:any processContents="lax" namespace="##other"/>
         <!-- (1,1) elements from (0,unbounded) namespaces -->
      </xs:choice>
   </xs:complexType>
   <xs:complexType name="lifeCycleType">
      <xs:sequence>
         <xs:element name="LifeCycleStates">
            <xs:complexType>
               <xs:sequence maxOccurs="unbounded">
                  <xs:element name="LifeCycleId" type="xs:ID"/>
                  <xs:element name="LifeCycleState">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:enumeration value="NOT_ISSUED"/>
                           <xs:enumeration value="ISSUED"/>
                           <xs:enumeration value="SUSPENDED"/>
                           <xs:enumeration value="REVOKED"/>
                           <xs:enumeration value="EXPIRED"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:element>
                  <xs:element name="LifeCycleTransitions">
                     <xs:complexType>
                        <xs:sequence maxOccurs="unbounded">
                           <xs:element name="lifeCycleTransition"
                              type="lifeCycleTransitionType"> </xs:element>
                        </xs:sequence>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="NumberOfStates" use="required" type="xs:integer"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="InitialState" use="required" type="xs:string" fixed="NOT_ISSUED"/>
   </xs:complexType>
   <xs:complexType name="lifeCycleStateType">
      <xs:sequence>
         <xs:element name="LifeCycleTransition" type="lifeCycleTransitionType"/>
      </xs:sequence>
      <xs:attribute name="LifeCycle" type="xs:ID"/>
      <xs:attribute name="LifeCycleStateName">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="NOT_ISSUED"/>
               <xs:enumeration value="ISSUED"/>
               <xs:enumeration value="SUSPENDED"/>
               <xs:enumeration value="REVOCKED"/>
               <xs:enumeration value="EXPIRED"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
   <xs:complexType name="lifeCycleTransitionType">
      <xs:sequence maxOccurs="1">
         <xs:element name="FromState">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="NOT_ISSUED"/>
                  <xs:enumeration value="ISSUED"/>
                  <xs:enumeration value="SUSPENDED"/>
                  <xs:enumeration value="REVOKED"/>
                  <xs:enumeration value="EXPIRED"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="ToState">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="NOT_ISSUED"/>
                  <xs:enumeration value="ISSUED"/>
                  <xs:enumeration value="SUSPENDED"/>
                  <xs:enumeration value="REVOKED"/>
                  <xs:enumeration value="EXPIRED"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="ConditionForLifeCycleTransition" type="lifeCycleConditionType"/>
      </xs:sequence>
```

```
</xs:complexType>
<xs:complexType name="lifeCycleConditionType">
  <xs:choice>
    <xs:element name="EvidenceIsValid" type="xs:boolean"/>
    <xs:element name="EvidenceIsNotValid" type="xs:boolean"/>
    <xs:element name="EvidenceIsQuestionable" type="xs:boolean"/>
    <xs:element name="ValidityPeriodExpired" type="xs:boolean"/>
  </xs:choice>
</xs:complexType>
<xs:attribute name="LifeCycleState">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="NOT_ISSUED"/>
      <xs:enumeration value="ISSUED"/>
      <xs:enumeration value="SUSPENDED"/>
      <xs:enumeration value="REVOKED"/>
      <xs:enumeration value="EXPIRED"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:complexType name="operativeConditionsType">
  <xs:sequence>
    <xs:element name="TocTechnicalSpecifications" type="tocTechSpecType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="propertyType">
  <xs:sequence>
    <xs:element name="propertyPerformance">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="propertyPerformanceRow" minOccurs="1"
            maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="propertyPerformanceCell" minOccurs="1"
                  maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:anySimpleType">
                       <xs:attribute name="name" type="xs:string"
                       use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="propertyParameterList">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="propertyParameter" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:anySimpleType">
                  <xs:attribute name="name" type="xs:string" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="class" type="xs:anyURI" use="required"/>
</xs:complexType>
<xs:element name="SecurityProperty" type="securityPropertyType"/>
<xs:complexType name="securityPropertyType">
  <xs:sequence maxOccurs="1" minOccurs="1">
    <xs:element name="sProperty" type="propertyType"/>
  </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>
<xs:complexType name="signatureType">
```

```
  <xs:sequence>
    <xs:element name="SignedInfo">
      <xs:complexType>
        <xs:sequence> </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="SignatureValue"/>
    <xs:element name="KeyInfo" type="KeyInfoType"/>
    <xs:element name="Object" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="qualifiedType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[_a-zA-Z][-_a-zA-Z0-9]*:[_a-zA-Z][-_a-zA-Z0-9]*"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="tocTechSpecType">
  <xs:sequence maxOccurs="1">
    <xs:element name="TocVendor" type="xs:string"/>
    <xs:element name="TocRelease" type="xs:string"/>
    <xs:element name="TocDate" type="xs:date" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="testMetricsType">
  <xs:sequence>
    <xs:element name="OperationCoverage"> </xs:element>
    <xs:element name="InputPartitionCoverage"/>
    <xs:element name="BranchCoverage"/>
    <xs:element name="ConditionCoverage"/>
    <xs:element name="PathCoverage" type="xs:string"/>
    <xs:element name="AttackCoverage"/>
    <xs:element name="Other"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="tocType">
  <xs:sequence>
    <xs:element name="CloudLayer" maxOccurs="unbounded">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="SaaS"/>
          <xs:enumeration value="PaaS"/>
          <xs:enumeration value="IaaS"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ConcreteToc" type="xs:string"> </xs:element>
    <xs:element name="TocDescription">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="WS"/>
          <xs:enumeration value="Application"/>
          <xs:enumeration value="DBMS"/>
          <xs:enumeration value="WEBSERVER"/>
          <xs:enumeration value="EMAIL"/>
          <xs:enumeration value="CRM"/>
          <xs:enumeration value="SDK"/>
          <xs:enumeration value="VIRTUALMACHINE"/>
          <xs:enumeration value="HD"/>
          <xs:enumeration value="SWITCH"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="TocURI" type="xs:anyURI" default="http://www.cumulus-project.eu"/>
    <xs:element name="ToTs" type="targetOfTestsType"/>
    <xs:sequence maxOccurs="unbounded" minOccurs="1">
      <xs:element name="OperativeCondition" type="operativeConditionsType"> </xs:element>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="Id" use="required" type="xs:ID"> </xs:attribute>
</xs:complexType>
<xs:complexType name="abstracCollectorType">
  <xs:sequence>
    <xs:element name="Aggregator" type="aggregatorType"/>
    <xs:element name="TestCategory">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Functionality"/>
          <xs:enumeration value="Robustness"/>
          <xs:enumeration value="ResilienceToAttacks"/>
          <xs:enumeration value="PenetrationTest"/>
```

```
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="Actionability" type="actionabilityType" minOccurs="0"/>
         <xs:element name="Capability" type="capabilityType" minOccurs="0"/>
         <xs:element name="TestType" type="xs:string"/>
         <xs:element name="TestDescription" type="xs:string"/>
         <xs:element name="TestGenerationModelLink" type="xs:anyURI"
            default="http://www.cumulus-project.eu"/>
         <xs:element name="TestAttributes" maxOccurs="1" minOccurs="1">
            <xs:complexType>
               <xs:sequence maxOccurs="unbounded">
                  <xs:element name="TestAttribute" type="testAttributeType"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="TestCases">
            <xs:complexType>
               <xs:sequence maxOccurs="unbounded">
                  <xs:element name="TestCase" type="testCaseType"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="Id" type="xs:ID" use="required"/>
   </xs:complexType>
   <xs:complexType name="testAttributeType">
      <xs:sequence>
         <xs:element name="ID" type="xs:integer"/>
         <xs:element name="Name" type="xs:string"/>
         <xs:element name="Value" type="xs:anyType"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="testCaseType">
      <xs:sequence maxOccurs="1">
         <xs:element name="ID" type="xs:integer"/>
         <xs:element name="Description" type="xs:string"/>
         <xs:element name="TestInstance" type="testInstanceType" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="testInstanceType">
      <xs:sequence>
         <xs:element name="Preconditions" type="xs:string"/>
         <xs:element name="HiddenCommunications" type="xs:string"/>
         <xs:element name="Input" type="xs:string"/>
         <xs:element name="ExpectedOutput" type="xs:string"/>
         <xs:element name="PostConditions" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="Operation" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:complexType name="targetOfTestsType">
      <xs:sequence maxOccurs="unbounded">
         <xs:element name="ToT" type="totType"> </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="totType">
      <xs:sequence maxOccurs="1">
         <xs:element name="Id" type="xs:integer"> </xs:element>
         <xs:element name="Target" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="testerType">
      <xs:sequence maxOccurs="1">
         <xs:element name="Name" type="xs:string"/>
         <xs:element name="Role" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="TC_Hash" type="xs:hexBinary"/>
   <xs:complexType name="TC_IntegrityProtectionType">
      <xs:sequence>
         <xs:element ref="TC_Hash"/>
         <xs:element name="HashAlgorithm" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="thresholdTrafficType">
      <xs:attribute name="TrafficType" type="xs:string" use="required"/>
      <xs:attribute name="ThresholdUnit" type="xs:string" use="required"/>
      <xs:attribute name="ThresholdValue" type="xs:float" use="required"/>
   </xs:complexType>
</xs:schema>
```

10.2. Test-based Certification Model instance (.xml file)

```
<?xml version="1.0" encoding="UTF-8"?>
<TestBasedCertificationModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/Users/filippogaudenzi/Documents/workspace/CumulusTestManager/XML_Repository/Cert
ificationModelv3.xsd">
  <CertificationModelId>
    <CmId>TEST000027</CmId>
  </CertificationModelId>
  <Collectors>
        <AbstractCollector Id="abstract1">
          <Aggregator >
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <TestType>Model Control Flow</TestType>
          <TestDescription>Send user/pass (with malformed pass) to registerUser API. RegisterUser result must be
"fail".</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model005.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>1</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
          <TestCase>
          <ID>1</ID>
          <Description>user generation</Description>
           <TestInstance Operation="registerUser">
            <Preconditions/>
            <HiddenCommunications/>
            <Input>username=fred; password="12345"</Input>
            <ExpectedOutput>fail</ExpectedOutput>
            <PostConditions></PostConditions>
           </TestInstance>
           </TestCase>
          </TestCases>
        </AbstractCollector>
        <AbstractCollector Id="abastract2">
          <Aggregator>
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
        <TestType></TestType>
          <TestDescription>Send user/pass (with wellformed pass) to registerUser API. Login result must be
"success".</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model006.html</TestGenerationModelLink>
        <TestAttributes>
          <TestAttribute>
            <ID>1</ID>
            <Name>cardinality</Name>
            <Value>1</Value>
          </TestAttribute>
        </TestAttributes>
        <TestCases>
          <TestCase>
            <ID>1</ID>
            <Description>login</Description>
            <TestInstance Operation="login">
              <Preconditions>ACL with user/pass with password satisfying secret-level1-policy-correctness (n. of pairs =
1)</Preconditions>
              <HiddenCommunications></HiddenCommunications>
              <Input>username,password</Input>
              <ExpectedOutput>success</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
        </TestCases>
      </AbstractCollector>
        <AbstractCollector Id="abstract3">
          <Aggregator>
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
        <TestType></TestType>
        <TestDescription>Send user/pass (with wellformed pass) to login API for 100 times. Logins must fails. 100th login  must
lockout user.</TestDescription>
        <TestGenerationModelLink>http://www.cumulus-project.eu/model007.html</TestGenerationModelLink>
        <TestAttributes>
          <TestAttribute>
            <ID>1</ID>
            <Name>cardinality</Name>
            <Value>100</Value>
          </TestAttribute>
        </TestAttributes>
        <TestCases>
          <TestCase>
            <ID>1</ID>
            <Description></Description>
            <TestInstance Operation="login">
              <Preconditions>sleepTime="00:07:00.000"</Preconditions>
              <HiddenCommunications/>
              <Input>username,password</Input>
              <ExpectedOutput>fail</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
          <TestCase>
            <ID>2</ID>
            <Description></Description>
            <TestInstance Operation="login">
              <Preconditions>sleepTime="00:07:00.000"</Preconditions>
              <HiddenCommunications/>
              <Input>username,password</Input>
              <ExpectedOutput>fail</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
          <TestCase>
            <ID>3</ID>
            <Description></Description>
            <TestInstance Operation="login">
              <Preconditions>sleepTime="00:07:00.000"</Preconditions>
              <HiddenCommunications/>
              <Input>username,password</Input>
              <ExpectedOutput>fail</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
          <TestCase>
            <ID>4</ID>
            <Description></Description>
            <TestInstance Operation="login">
              <Preconditions>sleepTime="00:07:00.000"</Preconditions>
              <HiddenCommunications/>
              <Input>username,password</Input>
              <ExpectedOutput>fail</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
          <TestCase>
            <ID>5</ID>
            <Description></Description>
            <TestInstance Operation="login">
              <Preconditions>sleepTime="00:07:00.000"</Preconditions>
              <HiddenCommunications/>
              <Input>username,password</Input>
              <ExpectedOutput>lockout</ExpectedOutput>
              <PostConditions></PostConditions>
            </TestInstance>
          </TestCase>
        </TestCases>
      </AbstractCollector>
        <AbstractCollector Id="abstract4">
          <Aggregator>
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <Actionability>
```

```
            <Description>mechanism</Description>
            <Mechanism>access control</Mechanism>
            <MechanismURI>http://www.cumulus.eu/mechanism1033.html</MechanismURI>
          </Actionability>
          <Capability>
            <Attacker>CloudProvider</Attacker>
            <AttackName></AttackName>
            <AttackerCapabilities>
              <CapabilityId>1</CapabilityId>
              <AttackerCapability>R</AttackerCapability>
              <CapabilityId>2</CapabilityId>
              <AttackerCapability>W</AttackerCapability>
              <CapabilityId>3</CapabilityId>
              <AttackerCapability>X</AttackerCapability>
            </AttackerCapabilities>
            <Attacker>Hacker</Attacker>
            <AttackName></AttackName>
            <AttackerCapabilities>
              <CapabilityId>1</CapabilityId>
              <AttackerCapability>R</AttackerCapability>
            </AttackerCapabilities>
          </Capability>
          <TestType></TestType>
          <TestDescription>Send user/pass (with malformed pass) to registerUser API. RegisterUser result must be
"fail".</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model005.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>1</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
            <TestCase>
              <ID>1</ID>
              <Description>user generation</Description>
              <TestInstance Operation="registerUser">
                <Preconditions/>
                <HiddenCommunications/>
                <Input>username=fred; password="1234567"</Input>
                <ExpectedOutput>fail</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
          </TestCases>
        </AbstractCollector>
        <AbstractCollector Id="abstract5">
          <Aggregator>
            <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
            <TestMetric>
              <OperationCoverage>1</OperationCoverage>
              <InputPartitionCoverage/>
              <BranchCoverage/>
              <ConditionCoverage/>
              <PathCoverage/>
              <AttackCoverage/>
              <Other> </Other>
            </TestMetric>
            <ElementForExtension>
              <Environment/>
              <TestingTool>no</TestingTool>
              <Code/>
              <Others/>
            </ElementForExtension>
          </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <TestType></TestType>
          <TestDescription>Send user/pass (with wellformed pass) to registerUser API. Login result must be
"success".</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model006.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>1</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
            <TestCase>
              <ID>1</ID>
              <Description>login</Description>
              <TestInstance Operation="login">
                <Preconditions></Preconditions>
                <HiddenCommunications></HiddenCommunications>
                <Input>username,password</Input>
                <ExpectedOutput>success</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
          </TestCases>
        </AbstractCollector>
    <AbstractCollector Id="abstract6">
      <Aggregator>
        <ModelLink>http://www.cumulus-project.eu/Model005.html</ModelLink>
        <TestMetric>
          <OperationCoverage>1</OperationCoverage>
          <InputPartitionCoverage/>
          <BranchCoverage/>
          <ConditionCoverage/>
          <PathCoverage/>
          <AttackCoverage/>
          <Other> </Other>
        </TestMetric>
        <ElementForExtension>
          <Environment/>
          <TestingTool>no</TestingTool>
          <Code/>
          <Others/>
        </ElementForExtension>
      </Aggregator>
          <TestCategory>Functionality</TestCategory>
          <TestType></TestType>
          <TestDescription>Send user/pass (with wellformed pass) to login API for 100 times. Logins must fails. 100th login
must lockout user.</TestDescription>
          <TestGenerationModelLink>http://www.cumulus-project.eu/model007.html</TestGenerationModelLink>
          <TestAttributes>
            <TestAttribute>
              <ID>1</ID>
              <Name>cardinality</Name>
              <Value>100</Value>
            </TestAttribute>
          </TestAttributes>
          <TestCases>
            <TestCase>
              <ID>1</ID>
              <Description></Description>
              <TestInstance Operation="login">
                <Preconditions>sleepTime="00:07:00.000"</Preconditions>
                <HiddenCommunications/>
                <Input>username,password</Input>
                <ExpectedOutput>fail</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
            <TestCase>
              <ID>2</ID>
              <Description></Description>
              <TestInstance Operation="login">
                <Preconditions>sleepTime="00:07:00.000"</Preconditions>
                <HiddenCommunications/>
                <Input>username,password</Input>
                <ExpectedOutput>fail</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
            <TestCase>
              <ID>3</ID>
              <Description></Description>
              <TestInstance Operation="login">
                <Preconditions>sleepTime="00:07:00.000"</Preconditions>
                <HiddenCommunications/>
                <Input>username,password</Input>
                <ExpectedOutput>fail</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
            <TestCase>
              <ID>4</ID>
              <Description></Description>
              <TestInstance Operation="login">
                <Preconditions>sleepTime="00:07:00.000"</Preconditions>
                <HiddenCommunications/>
                <Input>username,password</Input>
                <ExpectedOutput>fail</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
            <TestCase>
              <ID>101</ID>
              <Description></Description>
              <TestInstance Operation="login">
                <Preconditions>sleepTime="00:07:00.000"</Preconditions>
                <HiddenCommunications/>
                <Input>username,password</Input>
                <ExpectedOutput>lockout</ExpectedOutput>
                <PostConditions></PostConditions>
              </TestInstance>
            </TestCase>
          </TestCases>
        </AbstractCollector>
    <Collector ExpirationTime="2014-09" toDeploy="true" isStatic="true" Id="col2">
<ConditionForSomministration><DeltaTime>12:00:00</DeltaTime></ConditionForSomministration>
      <AbstractCollector>abstract1</AbstractCollector>
    </Collector>
    <Collector ExpirationTime="2014-09"  toDeploy="true" isStatic="true" Id="col1">
      <ConditionForSomministration>
        <Event>
          <Action>Login</Action>
          <Condition> failed
          </Condition>
          <Value>100</Value>
        </Event>
      </ConditionForSomministration>
      <AbstractCollector>abstract4</AbstractCollector>
    </Collector>
  </Collectors>
  <LifeCycle InitialState="NOT_ISSUED">
    <LifeCycleStates NumberOfStates="5">
      <LifeCycleId>ID1</LifeCycleId>
      <LifeCycleState>NOT_ISSUED</LifeCycleState>
      <LifeCycleTransitions>
        <lifeCycleTransition>
          <FromState>NOT_ISSUED</FromState>
          <ToState>ISSUED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsValid>true</EvidenceIsValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
      </LifeCycleTransitions>
      <LifeCycleId>ID2</LifeCycleId>
      <LifeCycleState>ISSUED</LifeCycleState>
      <LifeCycleTransitions>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>EXPIRED</ToState>
          <ConditionForLifeCycleTransition>
            <ValidityPeriodExpired>true</ValidityPeriodExpired>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>SUSPENDED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsQuestionable>true</EvidenceIsQuestionable>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>REVOKED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsNotValid>true</EvidenceIsNotValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>ISSUED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsValid>true</EvidenceIsValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
      </LifeCycleTransitions>
      <LifeCycleId>ID3</LifeCycleId>
      <LifeCycleState>SUSPENDED</LifeCycleState>
      <LifeCycleTransitions>
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>EXPIRED</ToState>
          <ConditionForLifeCycleTransition>
            <ValidityPeriodExpired>true</ValidityPeriodExpired>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>ISSUED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsValid>true</EvidenceIsValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>REVOKED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsNotValid>true</EvidenceIsNotValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>SUSPENDED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsQuestionable>true</EvidenceIsQuestionable>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
      </LifeCycleTransitions>
      <LifeCycleId>ID4</LifeCycleId>
      <LifeCycleState>EXPIRED</LifeCycleState>
      <LifeCycleTransitions>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>EXPIRED</ToState>
          <ConditionForLifeCycleTransition>
            <ValidityPeriodExpired>true</ValidityPeriodExpired>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
```

```
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>EXPIRED</ToState>
          <ConditionForLifeCycleTransition>
            <ValidityPeriodExpired>true</ValidityPeriodExpired>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
      </LifeCycleTransitions>
      <LifeCycleId>ID5</LifeCycleId>
      <LifeCycleState>REVOKED</LifeCycleState>
      <LifeCycleTransitions>
        <lifeCycleTransition>
          <FromState>ISSUED</FromState>
          <ToState>REVOKED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsNotValid>true</EvidenceIsNotValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
        <lifeCycleTransition>
          <FromState>SUSPENDED</FromState>
          <ToState>REVOKED</ToState>
          <ConditionForLifeCycleTransition>
            <EvidenceIsNotValid>true</EvidenceIsNotValid>
          </ConditionForLifeCycleTransition>
        </lifeCycleTransition>
      </LifeCycleTransitions>
    </LifeCycleStates>
  </LifeCycle>
  <Toc Id="ID001">
    <CloudLayer>SaaS</CloudLayer>
    <ConcreteToc>e-health v1.0</ConcreteToc>
    <TocDescription>Application</TocDescription>
    <TocURI>10.0.0.155</TocURI>
    <ToTs>
      <ToT>
        <Id>0</Id>
        <Target>Target0</Target>
      </ToT>
      <ToT>
        <Id>0</Id>
        <Target>Target1</Target>
      </ToT>
    </ToTs>
    <OperativeCondition>
      <TocTechnicalSpecifications>
        <TocVendor>ATOS</TocVendor>
        <TocRelease>1.0</TocRelease>
        <TocDate>2014-09-24</TocDate>
      </TocTechnicalSpecifications>
    </OperativeCondition>
  </Toc>
  <SecurityProperty SecurityPropertyId="Id101" SecurityPropertyDefinition="This property measures the strength of the
mechanism used to authenticate a user, on a scale from 0 to 4, notably taking into account identity proofing, credential security
during transfer and storage.">
    <sProperty class="http://cumulus-project.eu/security-properties#IAM:identity-assurance:user-authentication-and-identity-
assurance-level">
      <propertyPerformance>
        <propertyPerformanceRow>
          <propertyPerformanceCell name="level">1</propertyPerformanceCell>
        </propertyPerformanceRow>
      </propertyPerformance>
      <propertyParameterList>
      </propertyParameterList>
    </sProperty>
  </SecurityProperty>
  <Signature>
    <Name>SesarLab</Name>
    Role>Laboratory</Role>
  </Signature>
 <Context/>
</TestBasedCertificationModel>
```

10.3. Monitoring-based Certification Model generic schema (.xsd file)

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema xmlns:slasoi="http://www.slaatsoi.eu/slamodel" elementFormDefault="qualified"
targetNamespace="http://www.slaatsoi.eu/slamodel" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SLATemplate">
    <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:extension base="slasoi:AnnotatedType">
          <xs:sequence>
            <xs:element minOccurs="1" maxOccurs="1" name="UUID" type="slasoi:UUIDType" />
            <xs:element minOccurs="1" maxOccurs="1" name="ModelVersion" type="xs:string" />
            <xs:element minOccurs="1" maxOccurs="unbounded" name="Party" type="slasoi:AgreementPartyType" />
            <xs:element minOccurs="1" maxOccurs="unbounded" name="InterfaceDeclr" type="slasoi:InterfaceDeclrType" />
            <xs:element minOccurs="0" maxOccurs="unbounded" name="VariableDeclr" type="slasoi:VariableDeclrType" />
            <xs:element minOccurs="1" maxOccurs="unbounded" name="AgreementTerm" type="slasoi:AgreementTermType" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Assertion">
    <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:extension base="slasoi:AnnotatedType">
          <xs:sequence>
            <xs:sequence>
              <xs:element minOccurs="1" maxOccurs="1" name="UUID" type="slasoi:UUIDType" />
              <xs:element minOccurs="1" maxOccurs="1" name="EffectiveFrom" type="slasoi:TimeType" />
              <xs:element minOccurs="1" maxOccurs="1" name="EffectiveUntil" type="slasoi:TimeType" />
              <xs:element minOccurs="1" maxOccurs="1" name="AgreedAt" type="slasoi:TimeType" />
              <xs:element minOccurs="1" maxOccurs="unbounded" name="Party" type="slasoi:AgreementPartyType" />
              <xs:element minOccurs="0" maxOccurs="unbounded" name="AbstractParty" type="slasoi:AbstractPartyType" />
              <xs:element minOccurs="1" maxOccurs="unbounded" name="InterfaceDeclr" type="slasoi:InterfaceDeclrType" />
              <xs:element minOccurs="0" maxOccurs="unbounded" name="VariableDeclr" type="slasoi:VariableDeclrType" />
              <xs:element minOccurs="1" maxOccurs="unbounded" name="AgreementTerm" type="slasoi:AgreementTermType" />
            </xs:sequence>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="AgreementPartyType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
          <xs:element minOccurs="0" maxOccurs="unbounded" name="Operative" type="slasoi:OperativeType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Role" type="slasoi:STNDType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="AbstractPartyType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Role" type="slasoi:STNDType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="OperativeType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="VariableDeclrType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:choice>
          <xs:sequence>
            <xs:element minOccurs="1" maxOccurs="1" name="Var" type="slasoi:IDType" />
            <xs:element minOccurs="1" maxOccurs="1" name="Expr" type="slasoi:ExprType" />
            <xs:element minOccurs="0" maxOccurs="unbounded" name="Initially" type="slasoi:ValueExprType"></xs:element>
          </xs:sequence>
          <xs:element minOccurs="1" name="Customisable" type="slasoi:CustomisableType" />
          <xs:element minOccurs="1" maxOccurs="1" name="serviceref" type="slasoi:IDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="List" type="slasoi:ListVarType"/>
        </xs:choice>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ListVarType">
    <xs:sequence>
      <xs:element name="Index" type="slasoi:IDType"></xs:element>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Value" type="slasoi:ValueExprType"></xs:element>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Domain" type="slasoi:DomainExprType"></xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CustomisableType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="Var" type="slasoi:IDType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Value" type="slasoi:CONSTType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Expr" type="slasoi:DomainExprType" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Initially" type="slasoi:ValueExprType"></xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AgreementTermType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
          <xs:element minOccurs="0" maxOccurs="1" name="Precondition" type="slasoi:ConstraintExprType" />
          <xs:element minOccurs="0" maxOccurs="unbounded" name="VariableDeclr" type="slasoi:VariableDeclrType" />
          <xs:element minOccurs="1" maxOccurs="unbounded" name="Guaranteed" type="slasoi:GuaranteedType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="GuaranteedType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:choice>
          <xs:element minOccurs="1" maxOccurs="1" name="State" type="slasoi:GuaranteedStateType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Action" type="slasoi:GuaranteedActionType" />
        </xs:choice>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="GuaranteedStateType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Priority" nillable="true" type="slasoi:CONSTType" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Constraint" type="slasoi:ConstraintExprType" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Precondition" type="slasoi:ConstraintExprType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="GuaranteedActionType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
      <xs:element minOccurs="1" maxOccurs="1" name="ActorRef" type="slasoi:IDType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Policy" type="slasoi:STNDType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Precondition" type="slasoi:EventExprType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Postcondition" type="slasoi:GuaranteedActionDefnType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="GuaranteedActionDefnType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:choice>
          <xs:element minOccurs="1" maxOccurs="1" name="Invocation_action" type="slasoi:InvocationActionType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Var_update_action" type="slasoi:VarUpdateActionType" />
        </xs:choice>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="VarUpdateActionType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="Expr" type="slasoi:ValueExprType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="InvocationActionType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="Endpoint" type="slasoi:IDType" />
      <xs:element minOccurs="1" maxOccurs="1" name="Operation" type="slasoi:IDType" />
      <xs:element minOccurs="0" maxOccurs="1" name="Parameters" type="slasoi:MapIdValueExpr" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="InterfaceDeclrType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="ProviderRef" type="slasoi:IDType" />
          <xs:element minOccurs="0" maxOccurs="unbounded" name="Endpoint" type="slasoi:EndpointType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Interface" type="slasoi:InterfaceType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="EndpointType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="ID" type="slasoi:IDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Location" type="slasoi:UUIDType" />
          <xs:element minOccurs="1" maxOccurs="1" name="Protocol" type="slasoi:STNDType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="InterfaceType">
    <xs:choice>
      <xs:element minOccurs="1" maxOccurs="1" name="InterfaceRef" type="slasoi:InterfaceRefType" />
      <xs:element minOccurs="1" maxOccurs="1" name="InterfaceSpec" type="slasoi:InterfaceSpecType" />
      <xs:element minOccurs="1" maxOccurs="1" name="InterfaceResourceType" type="slasoi:InterfaceResourceTypeType" />
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="InterfaceRefType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element name="InterfaceLocation" type="slasoi:UUIDType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="InterfaceResourceTypeType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="Name" type="xs:string" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="InterfaceSpecType">
    <xs:complexContent mixed="false">
      <xs:extension base="slasoi:AnnotatedType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="Name" type="xs:string" />
          <xs:element minOccurs="0" maxOccurs="unbounded" name="Extended" type="slasoi:IDType" />
          <xs:element minOccurs="0" maxOccurs="unbounded" name="Operation" type="slasoi:InterfaceOperationType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
```

```
</xs:complexType>
<xs:complexType name="InterfaceOperationType">
  <xs:complexContent mixed="false">
    <xs:extension base="slasoi:AnnotatedType">
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="1" name="Name" type="slasoi:IDType" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="Input" type="slasoi:InterfaceOperationPropertyType" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="Output" type="slasoi:InterfaceOperationPropertyType" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="Related" type="slasoi:InterfaceOperationPropertyType" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="Fault" type="slasoi:STNDType" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="InterfaceOperationPropertyType">
  <xs:complexContent mixed="false">
    <xs:extension base="slasoi:AnnotatedType">
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="1" name="Name" type="slasoi:IDType" />
        <xs:element minOccurs="1" maxOccurs="1" name="Auxiliary" type="xs:boolean" />
        <xs:element minOccurs="0" maxOccurs="1" name="Datatype" type="slasoi:STNDType" />
        <xs:element minOccurs="0" maxOccurs="1" name="Domain" type="slasoi:DomainExprType" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ServiceRefType">
  <xs:choice>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" name="InterfaceList" type="slasoi:IDListType" />
      <xs:element minOccurs="0" maxOccurs="1" name="OperationList" type="slasoi:IDListType" />
      <xs:element minOccurs="0" maxOccurs="1" name="EndpointList" type="slasoi:IDListType" />
    </xs:sequence>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="ServiceRef" type="slasoi:IDType"></xs:element>
    </xs:sequence>
  </xs:choice>

</xs:complexType>
<xs:complexType name="IDListType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="ID" type="slasoi:IDType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AnnotatedType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="Text" type="xs:string" />
    <xs:element minOccurs="1" maxOccurs="1" name="Properties" type="slasoi:MapStndAny" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MapStndAny">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="Entry" type="slasoi:StndAnyEntryType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MapIdValueExpr">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="Entry" type="slasoi:IdValueExprEntryType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="StndAnyEntryType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="Key" type="slasoi:STNDType" />
    <xs:element minOccurs="1" maxOccurs="1" name="Value" type="xs:anyType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="IdValueExprEntryType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="Key" type="slasoi:IDType" />
    <xs:element minOccurs="1" maxOccurs="1" name="Value" type="slasoi:ValueExprType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ConstraintExprType">
  <xs:choice>
    <xs:sequence>
      <xs:element minOccurs="0" name="ID" type="slasoi:IDType"/>
      <xs:element minOccurs="1" maxOccurs="1" name="ValueExpr" type="slasoi:ValueExprType" />
    </xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="CountExpr" type="slasoi:CountExprType"/>
    <xs:element minOccurs="1" maxOccurs="1" name="FuncExpr" type="slasoi:FuncExprType" />
    <xs:element minOccurs="1" maxOccurs="1" name="CompoundConstraintExpr" type="slasoi:CompoundConstraintExprType" />
    <xs:element minOccurs="1" maxOccurs="1" name="TypeConstraintExpr" type="slasoi:TypeConstraintExprType" />
  </xs:choice>
</xs:complexType>
<xs:complexType name="CountExprType">
  <xs:sequence>
    <xs:element minOccurs="0"  maxOccurs="unbounded" name="FuncExpr" type="slasoi:FuncExprType"/>
    <xs:element name="EventExpr" type="slasoi:EventExprType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="DomainExpr" type="slasoi:DomainExprType"></xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SeriesExprType">
  <xs:sequence>
    <xs:element minOccurs="0"  maxOccurs="unbounded" name="FuncExpr" type="slasoi:FuncExprType"/>
    <xs:element name="EventExpr" type="slasoi:EventExprType"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="CountExpr" type="slasoi:CountExprType"></xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SeriesVarType">
  <xs:sequence>
    <xs:element name="Series" type="slasoi:SeriesExprType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="TypeConstraintExprType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="Value" type="slasoi:ValueExprType" />
    <xs:element minOccurs="1" maxOccurs="1" name="Domain" type="slasoi:DomainExprType" />
    <xs:element minOccurs="0" maxOccurs="1" name="Error" type="slasoi:CONSTType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CompoundConstraintExprType">
  <xs:choice>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="Subexpression" type="slasoi:ConstraintExprType" />
      <xs:element minOccurs="1" maxOccurs="1" name="LogicalOp" type="slasoi:STNDType" />
    </xs:sequence>
  </xs:choice>
</xs:complexType>
<xs:complexType name="DomainExprType">
  <xs:choice>
    <xs:element minOccurs="1" maxOccurs="1" name="SimpleDomainExpr" type="slasoi:SimpleDomainExprType" />
    <xs:element minOccurs="1" maxOccurs="1" name="CompoundDomainExpr" type="slasoi:CompoundDomainExprType" />
    <xs:element name="PrimitiveDomainExpr">
      <xs:complexType>
        <xs:attribute name="type">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="integer" />
              <xs:enumeration value="float" />
              <xs:enumeration value="string" />
              <xs:enumeration value="boolean" />
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:complexType name="SimpleDomainExprType">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="ComparisonOp" type="slasoi:STNDType" />
    <xs:element minOccurs="1" maxOccurs="1" name="Value" type="slasoi:ValueExprType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CompoundDomainExprType">
  <xs:choice>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="Subexpression" type="slasoi:DomainExprType" />
      <xs:element minOccurs="1" maxOccurs="1" name="LogicalOp" type="slasoi:STNDType" />
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

```
<xs:complexType name="EventExprType">
 <xs:complexContent mixed="false">
  <xs:extension base="slasoi:AnnotatedType">
   <xs:choice>
    <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="1" name="Operator" type="slasoi:STNDType" />
     <xs:element minOccurs="0" maxOccurs="unbounded" name="Parameter" type="slasoi:ExprType" />
    </xs:sequence>
    <xs:choice>
     <xs:element name="Specialisation" type="slasoi:SpecialisationType"/>
     <xs:element name="Intersection" type="slasoi:IntersectionType"/>
     <xs:element name="Union" type="slasoi:UnionType"/>
     <xs:element name="Difference" type="slasoi:DifferenceType"/>
     <xs:element name="CallEvent" type="slasoi:CallEventType"/>
     <xs:element name="Periodic">
      <xs:complexType>
       <xs:attribute name="value" type="xs:string"/>
       <xs:attribute name="unit" type="xs:string"/>
      </xs:complexType>
     </xs:element>
     <xs:element name="Schedule">
      <xs:complexType>
       <xs:attribute name="time" type="xs:string"/>
      </xs:complexType>
     </xs:element>
     <xs:element name="EventTime">
      <xs:complexType>
       <xs:attribute name="time" type="xs:string"/>
      </xs:complexType>
     </xs:element>
     <xs:element name="Fault" type="slasoi:ServiceRefType"/>
     <xs:element name="Violated" type="slasoi:ConstraintExprType"/>
     <xs:element name="Warned">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="Expr" type="slasoi:ConstraintExprType"/>
        <xs:element name="Ration" type="xs:double"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="Recovered" type="slasoi:ConstraintExprType"/>
     <xs:element name="TimeOf" type="slasoi:TimeOfType"/>
     <xs:element name="Reply" type="slasoi:ReplyEventType"></xs:element>
    </xs:choice>
   </xs:choice>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="SpecialisationType">
 <xs:choice>
  <xs:sequence>
   <xs:element name="EventExpr" type="slasoi:EventExprType"/>
   <xs:element name="ConstraintExpr" type="slasoi:ConstraintExprType"/>
  </xs:sequence>
  <xs:sequence>
   <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
   <xs:element name="ConstraintExpr" type="slasoi:ConstraintExprType"/>
  </xs:sequence>
 </xs:choice>
</xs:complexType>

<xs:complexType name="IntersectionType">
 <xs:choice>
  <xs:sequence>
   <xs:element name="EventExpr1" type="slasoi:EventExprType"/>
   <xs:element name="EventExpr2" type="slasoi:EventExprType"/>
  </xs:sequence>
  <xs:sequence>
   <xs:element name="EventExpr" type="slasoi:EventExprType"/>
   <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
  </xs:sequence>
  <xs:sequence>
   <xs:element name="ValueExpr1" type="slasoi:ValueExprType"/>
   <xs:element name="ValueExpr2" type="slasoi:ValueExprType"/>
  </xs:sequence>
```

```
    <xs:sequence>
      <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
      <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>

<xs:complexType name="UnionType">
  <xs:choice>
    <xs:sequence>
      <xs:element name="EventExpr1" type="slasoi:EventExprType"/>
      <xs:element name="EventExpr2" type="slasoi:EventExprType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="EventExpr" type="slasoi:EventExprType"/>
      <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="ValueExpr1" type="slasoi:ValueExprType"/>
      <xs:element name="ValueExpr2" type="slasoi:ValueExprType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
      <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>

<xs:complexType name="DifferenceType">
  <xs:choice>
    <xs:sequence>
      <xs:element name="EventExpr1" type="slasoi:EventExprType"/>
      <xs:element name="EventExpr2" type="slasoi:EventExprType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="EventExpr" type="slasoi:EventExprType"/>
      <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="ValueExpr1" type="slasoi:ValueExprType"/>
      <xs:element name="ValueExpr2" type="slasoi:ValueExprType"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
      <xs:element name="EventExprVar" type="slasoi:EventExprVarType"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>

<xs:complexType name="EventExprVarType">
  <xs:sequence>
    <xs:element name="ID" type="slasoi:EventExprType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CallEventType">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="InvocationRef" type="slasoi:InvocationRefType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ReplyEventType">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="InvocationRef" type="slasoi:InvocationRefType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="TimeOfType">
  <xs:choice>
    <xs:element name="ValueExpr" type="slasoi:ValueExprType" />
    <xs:element name="EventExpr" type="slasoi:EventExprType" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="InvocationRefType">
  <xs:choice>
  <xs:element name="Service" type="slasoi:ServiceRefType"/>
  <xs:element name="Invocation" type="slasoi:InvocationActionType"/>
 </xs:choice>
</xs:complexType>

<xs:complexType name="FuncExprType">
 <xs:complexContent mixed="false">
  <xs:extension base="slasoi:AnnotatedType">
   <xs:choice>
    <xs:element name="FuncOp">
     <xs:complexType>
      <xs:choice>
       <xs:element name="ID" type="slasoi:IDType"/>
       <xs:element name="ArithmeticOp" type="slasoi:ArithmeticOpType"/>
       <xs:element name="SetOp" type="slasoi:SetOpType"/>
       <xs:element name="QosTerm" type="slasoi:QosTermType"/>
       <xs:element name="ListOp" type="slasoi:ListOpType"/>
       <xs:element name="TimeSeriesOp" type="slasoi:TimeSeriesOpType"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:sequence>
     <xs:element name="Operator" type="slasoi:STNDType" />
     <xs:element minOccurs="0" maxOccurs="unbounded" name="Parameter" type="slasoi:ValueExprType" />
    </xs:sequence>
   </xs:choice>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="ArithmeticOpType">
 <xs:choice>
  <xs:element name="Add" type="slasoi:OrderedArithmeticOperators"/>
  <xs:element name="Substract" type="slasoi:OrderedArithmeticOperators"/>
  <xs:element name="Multiply" type="slasoi:OrderedArithmeticOperators"/>
  <xs:element name="Divide" type="slasoi:OrderedArithmeticOperators"/>
  <xs:element name="Modulo" type="slasoi:OrderedArithmeticOperators"/>
  <xs:element name="Round" type="slasoi:OrderedArithmeticOperators"/>
 </xs:choice>
</xs:complexType>

<xs:complexType name="OrderedArithmeticOperators">
 <xs:choice>
  <xs:sequence>
   <xs:element name="ValueExpr1" type="slasoi:ValueExprType"/>
   <xs:element name="ValueExpr2" type="slasoi:ValueExprType"/>
  </xs:sequence>
  <xs:sequence>
   <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
   <xs:element name="ArithmeticOp" type="slasoi:ArithmeticOpType"/>
  </xs:sequence>
 </xs:choice>
</xs:complexType>

<xs:complexType name="SetOpType">
 <xs:choice>
  <xs:element name="Sum" type="slasoi:OrderedSetOperatorType"/>
  <xs:element name="Std" type="slasoi:OrderedSetOperatorType"/>
  <xs:element name="Mean" type="slasoi:OrderedSetOperatorType"/>
  <xs:element name="Mode" type="slasoi:OrderedSetOperatorType"/>
  <xs:element name="Max" type="slasoi:OrderedSetOperatorType"/>
  <xs:element name="Min" type="slasoi:OrderedSetOperatorType"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="OrderedSetOperatorType">
 <xs:sequence>
  <xs:element name="SeriesVar" type="slasoi:SeriesVarType"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="TimeSeriesOpType">
 <xs:choice>
  <xs:element name="Series" type="slasoi:SeriesExprType"/>
  <xs:element name="SeriesValue">
   <xs:complexType>
    <xs:sequence>
```

```
     <xs:element name="FuncExpr" type="slasoi:FuncExprType"/>
     <xs:element name="EventExpr" type="slasoi:EventExprType"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:choice>
</xs:complexType>

<xs:complexType name="ListOpType">
 <xs:choice>
  <xs:element name="Insert" type="slasoi:OrderedListOperatorType"/>
  <xs:element name="Remove" type="slasoi:OrderedListOperatorType"/>
  <xs:element name="Update" type="slasoi:OrderedListOperatorType"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="OrderedListOperatorType">
 <xs:sequence>
  <xs:element minOccurs="0" maxOccurs="unbounded" name="ID" type="slasoi:IDType"/>
  <xs:element minOccurs="0" maxOccurs="unbounded" name="ListVar" type="slasoi:ListVarType"/>
  <xs:element name="ConstraintExpr" type="slasoi:ConstraintExprType"/>
  <xs:element name="ValueExpr" type="slasoi:ValueExprType"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="QosTermType">
 <xs:choice>
  <xs:element name="Availability" type="slasoi:OrderedQoSTermType"/>
  <xs:element name="NonRepudiation" type="slasoi:OrderedQoSTermType"/>
  <xs:element name="ArrivalRate" type="slasoi:OrderedQoSTermType"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="OrderedQoSTermType">
 <xs:sequence>
  <xs:element name="ServiceVar" type="slasoi:ServiceRefType"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ExprType">
 <xs:choice>
  <xs:element name="ValueExpr" type="slasoi:ValueExprType" />
  <xs:element name="ConstraintExpr" type="slasoi:ConstraintExprType" />
 </xs:choice>
</xs:complexType>
<xs:complexType name="ValueExprType">
 <xs:choice>
  <xs:element name="ID" type="slasoi:IDType" />
  <xs:element name="BOOL" type="slasoi:BoolType" />
  <xs:element name="CONST" type="slasoi:CONSTType" />
  <xs:element name="TIME" type="slasoi:TimeType" />
  <xs:element name="PATH" type="slasoi:PathType" />
  <xs:element name="UUID" type="slasoi:UUIDType" />
  <xs:element name="STND" type="slasoi:STNDType" />
  <xs:element name="FuncExpr" type="slasoi:FuncExprType" />
  <xs:element name="EventExpr" type="slasoi:EventExprType" />
  <xs:element name="DomainExpr" type="slasoi:DomainExprType" />
  <xs:element name="ServiceRef" type="slasoi:ServiceRefType" />
  <xs:element name="ListValueExpr" type="slasoi:ListValueExprType" />
 </xs:choice>
</xs:complexType>
<xs:simpleType name="UUIDType">
 <xs:union memberTypes="xs:anyURI" />
</xs:simpleType>
<xs:simpleType name="STNDType">
 <xs:union memberTypes="slasoi:UUIDType" />
</xs:simpleType>
<xs:simpleType name="TimeType">
 <xs:union memberTypes="xs:dateTime" />
</xs:simpleType>
<xs:simpleType name="BoolType">
 <xs:union memberTypes="xs:boolean" />
</xs:simpleType>
<xs:simpleType name="IDType">
 <xs:union memberTypes="slasoi:PathType slasoi:UUIDType">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z0-9\-._~!$&amp;'\(\)\*\+,;=:@%/?]*" />
    <xs:whiteSpace value="collapse" />
```

```
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
  <xs:simpleType name="PathType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Za-z0-9][A-Za-z0-9\-._~!$&amp;'\(\)\*\+,;=:@%/?]*/|#[A-Za-z0-9][A-Za-z0-9\-
._~!$&amp;'\(\)\*\+,;=:@%/?]*" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="CONSTType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="Value" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="Datatype" type="slasoi:STNDType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ListValueExprType">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="Value" type="slasoi:ValueExprType" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Party" type="slasoi:AgreementPartyType" />
  <xs:element name="ModelVersion" type="xs:string" />
  <xs:element name="Annotated" type="slasoi:AnnotatedType" />
  <xs:element name="InterfaceDeclr" type="slasoi:InterfaceDeclrType" />
  <xs:element name="DomainExpr" type="slasoi:DomainExprType" />
  <xs:element name="EventExpr" type="slasoi:EventExprType" />
  <xs:element name="FuncExpr" type="slasoi:FuncExprType" />
  <xs:element name="ConstraintExpr" type="slasoi:ConstraintExprType" />
  <xs:element name="VariableDeclr" type="slasoi:VariableDeclrType" />
  <xs:element name="AgreementTerm" type="slasoi:AgreementTermType" />
  <xs:element name="InterfaceOperation" type="slasoi:InterfaceOperationType" />
  <xs:element name="Related" type="slasoi:InterfaceOperationPropertyType" />
  <xs:element name="EffectiveFrom" type="slasoi:TimeType" />
  <xs:element name="EffectiveUntil" type="slasoi:TimeType" />
  <xs:element name="TemplateId" type="slasoi:UUIDType" />
  <xs:element name="AgreedAt" type="slasoi:TimeType" />
</xs:schema>
```

10.4. Monitoring-based Certification Model instance (.xml file)

```
<?xml version="1.0" encoding="UTF-8"?>
<model:CertificationModel xmlns:sch="http://www.ascc.net/xml/schematron"
 xmlns:ec="http://slasoi.org/monitoring/citymonitor/xmlrule"
 xmlns:sla="http://www.slaatsoi.eu/slamodel"
 xmlns:model="http://www.cumulus.org/certificate/model"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.cumulus.org/certificate/model
file:/C:/Users/abfc152/Dropbox/Deliverables/schemas/CertificationModel_XMLSchema_v4.xsd">

  <Model_Id>1001</Model_Id>
  <CASignature>CUMULUS_City</CASignature>

  <TOC id="id">
    <providesInterface>
      <Text> </Text>
      <Properties>
      </Properties>
      <ID>interface::id::c::1/</ID>
      <ProviderRef>nr::id::c</ProviderRef>
      <Endpoint>
        <Text> </Text>
        <Properties> </Properties>
        <ID>c111</ID>
        <Location>http://www.cumulus-project.eu</Location>
        <Protocol> SOAP </Protocol>
      </Endpoint>
      <Interface>
        <InterfaceSpec>
          <Text> </Text>
        <Properties>  </Properties>
        <Name>nr::id::c::cloudinterface</Name>
        <Operation>
          <Text>rqsac</Text>
          <Properties> </Properties>
          <Name>data</Name>
        </Operation>
      </InterfaceSpec>
    </Interface>
  </providesInterface>
  <requiresInterface>
    <Text> </Text>
    <Properties></Properties>
    <ID> interface::id::d::1</ID>
    <ProviderRef></ProviderRef>
    <Endpoint>
      <Text> </Text>
      <Properties></Properties>
      <ID>d111</ID>
      <Location> http://www.cumulus-project.eu </Location>
      <Protocol> SOAP</Protocol>
    </Endpoint>
    <Interface>
      <InterfaceRef>
        <Text> rqsbc</Text>
        <Properties> </Properties>
        <InterfaceLocation> http://www.cumulus-project.eu </InterfaceLocation>
      </InterfaceRef>
    </Interface>
  </requiresInterface>

</TOC>

<SecurityProperty  Property_Category="AIS:non-repudiation:non-repudiation-of-origin”>
  <Assertion>
    <Text></Text>
    <Properties>
    </Properties>
    <UUID>url2 </UUID>
    <EffectiveFrom>00:00:00:00</EffectiveFrom>
    <EffectiveUntil>23:59:59:59</EffectiveUntil>
    <AgreedAt></AgreedAt>
    <AbstractParty>
      <Text></Text>
      <Properties>
      </Properties>
      <ID>c</ID>
      <Role>cloudprovider</Role>
    </AbstractParty>
    <AbstractParty>
      <Text></Text>
      <Properties>
      </Properties>
      <ID>a</ID>
      <Role>dataowner</Role>
    </AbstractParty>
    <AbstractParty>
      <Text></Text>
      <Properties>
      </Properties>
      <ID>b</ID>
      <Role>datauser</Role>
    </AbstractParty>
    <AbstractParty>
      <Text></Text>
      <Properties>
      </Properties>
```

```
   <ID>t</ID>
   <Role>trustedthirdparty</Role>
</AbstractParty>

<InterfaceDeclr>
   <Text></Text>
   <Properties>
   </Properties>
   <ID>c1</ID>
   <ProviderRef>c</ProviderRef>
   <Interface>
      <InterfaceSpec>
         <Text></Text>
         <Properties>
         </Properties>
         <Name>cloudinterface</Name>
         <Operation>
            <Text></Text>
            <Properties></Properties>
            <Name>rqsac</Name>
            <Input>
               <Name>data</Name>
               <Auxiliary>true</Auxiliary>
               <Datatype>url</Datatype>
               <Domain>0</Domain>
            </Input>
            <Input>
               <Text></Text>
               <Properties></Properties>
               <Name>t</Name>
               <Auxiliary>true</Auxiliary>
               <Datatype>url</Datatype>
               <Domain>0</Domain>
            </Input>
         </Operation>
         <Operation>
            <Text></Text>
            <Properties></Properties>
            <Name>rqsbc </Name>
            <Input>
               <Name>data</Name>
               <Auxiliary>true</Auxiliary>
               <Datatype>url</Datatype>
               <Domain>0</Domain>
            </Input>
         </Operation>
         <Operation>
            <Text></Text>
            <Properties></Properties>
            <Name>rqstc </Name>
            <Input>
               <Name>data</Name>
               <Auxiliary>true</Auxiliary>
               <Datatype>url</Datatype>
               <Domain>0</Domain>
            </Input>
         </Operation>
      </InterfaceSpec>
   </Interface>
</InterfaceDeclr>
<InterfaceDeclr>
   <Text></Text>
   <Properties>
   </Properties>
   <ID>a1</ID>
   <ProviderRef>a</ProviderRef>
   <Interface>
```

```
      <InterfaceSpec>
         <Text></Text>
         <Properties>
         </Properties>
         <Name>ainterface</Name>
         <Operation>
            <Input>
               < Text></ Text>
               < Properties>
               </ Properties>
               < Name>RSPca</ Name>
               < Auxiliary> true </ Auxiliary>
               < Datatype>url</ Datatype>
               <Domain></Domain>
            </Input>
         </Operation>
      </InterfaceSpec>
   </Interface>
</InterfaceDeclr>
<InterfaceDeclr>
   <Text></Text>
   <Properties>
   </Properties>
   <ID>b1</ID>
   <ProviderRef>b</ProviderRef>
   <Interface>
      <InterfaceSpec>
         <Text></Text>
         <Properties>
         </Properties>
         <Name>binterface</Name>
         <Operation>
            <Input>
               < Text></ Text>
               < Properties>
               </ Properties>
               < Name>RSPcb</ Name>
               < Auxiliary> true </ Auxiliary>
               < Datatype>url</ Datatype>
               <Domain></Domain>
            </Input>
         </Operation>
      </InterfaceSpec>
   </Interface>
</InterfaceDeclr>
<InterfaceDeclr>
   <Text></Text>
   <Properties>
   </Properties>
   <ID>t1</ID>
   <ProviderRef>t</ProviderRef>
   <Interface>
      <InterfaceSpec>
         <Text></Text>
         <Properties>
         </Properties>
         <Name>tinterface</Name>
         <Operation>
            <Input>
               < Text></ Text>
               < Properties>
               </ Properties>
               < Name>RSPct</ Name>
               < Auxiliary> true </ Auxiliary>
               < Datatype>url</ Datatype>
               <Domain></Domain>
            </Input>
         </Operation>
      </InterfaceSpec>
   </Interface>
</InterfaceDeclr>

<VariableDeclr>
   <Text/>
   <Properties/>
   <Var>rqsacv</Var>
   <Expr>
      <ValueExpr>
         <EventExpr>
            <Text></Text>
            <Properties></Properties>
            <Operator>invocation</Operator>
            <Parameter>
               <ValueExpr>
                  <ServiceRef>
                     <OperationList>
                        <ID>nroac</ID>
                     </OperationList>
                     <EndpointList>
                        <ID>c111</ID>
                     </EndpointList>
                  </ServiceRef>
               </ValueExpr>
            </Parameter>
         </EventExpr>
      </ValueExpr>
   </Expr>
</VariableDeclr>

<AgreementTerm>
   <Text></Text>
   <Properties></Properties>
   <ID>term1</ID>
   <Precondition>
      <CountExpr>
         <EventExpr>
            <Text></Text>
            <Properties></Properties>
            <Difference>
               <ValueExpr1>
                  <ListValueExpr>
                     <Value>rqsac</Value>
                  </ListValueExpr>
               </ValueExpr1>
               <ValueExpr2>
                  <ListValueExpr>
                     <Value>rqsacv</Value>
                  </ListValueExpr>
               </ValueExpr2>
            </Difference>
         </EventExpr>
         <DomainExpr></DomainExpr>
      </CountExpr>
   </Precondition>
   <Guaranteed>
      <Text></Text>
      <Properties></Properties>
      <State>
         <ID>gstate1</ID>
         <Constraint>
            <FuncExpr>
               <Text></Text>
               <Properties></Properties>
               <Operator>http://www.slaatsoi.org/coremodel#equals</Operator>
```

```
                            <Parameter>
                                <ID>rspcav</ID>
                            </Parameter>
                            <Parameter>
                                <ID>rqsacv</ID>
                            </Parameter>
                        </FuncExpr>
                    </Constraint>
                    <Constraint>
                        <CountExpr>
                            <EventExpr>
                                <Text></Text>
                                <Properties></Properties>
                                <Operator>http://www.slaatsoi.org/coremodel#less_than_or_equal</Operator>
                                <Parameter>
                                    <ConstraintExpr>
                                        <CountExpr>
                                            <EventExpr>
                                                <Text></Text>
                                                <Properties></Properties>
                                                <TimeOf>
                                                    <ValueExpr>
                                                        <ID>rspca</ID>
                                                    </ValueExpr>
                                                </TimeOf>
                                            </EventExpr>
                                        </CountExpr>
                                    </ConstraintExpr>
                                </Parameter>
                                <Parameter>
                                    <ConstraintExpr>
                                        <FuncExpr>
                                            <Text></Text>
                                            <Properties></Properties>
                                            <FuncOp>
                                                <ArithmeticOp>
                                                    <Add>
                                                        <ValueExpr1>
                                                            <EventExpr>
                                                                <Text></Text>
                                                                <Properties></Properties>
                                                                <TimeOf>
                                                                    <ValueExpr>
                                                                        <ID>rqsacv</ID>
                                                                    </ValueExpr>
                                                                </TimeOf>
                                                            </EventExpr>
                                                        </ValueExpr1>
                                                        <ValueExpr2>
                                                            <ID>t</ID>
                                                        </ValueExpr2>
                                                    </Add>
                                                </ArithmeticOp>
                                            </FuncOp>
                                        </FuncExpr>
                                    </ConstraintExpr>
                                </Parameter>
                            </EventExpr>
                        </CountExpr>
                    </Constraint>
                </State>
            </Guaranteed>
        </AgreementTerm>
    </Assertion>
</SecurityProperty>

<AssessmentScheme>
```

```
<EvidenceSufficiencyCondition Id="1011">
  <MonitoringPeriodCondition minMonitoredPeriod="720" periodUnit="hours"/>
</EvidenceSufficiencyCondition>
<ExpirationCondition Id="987">
  <elapsedPeriod period="1" periodUnit="years"/>
</ExpirationCondition>
<Conflict conflictId="1100" assertionId="guarantee1" assessmentPeriod="1" assessmentUnit=”month”/>

  <Anomalies>
    <Assertion Id="2103">
      <Text></Text>
      <Properties>
      </Properties>
      <UUID>url1</UUID>
      <EffectiveFrom>00:00:00:00</EffectiveFrom>
      <EffectiveUntil>23:59:59:59</EffectiveUntil>
      <AgreedAt></AgreedAt>
      <AbstractParty>
        [...]
      </AbstractParty>
      <InterfaceDeclr>
        [...]
      </InterfaceDeclr>
      <VariableDeclr>
        [...]
      </VariableDeclr>
      <VariableDeclr>
        <Text></Text>
        <Properties></Properties>
        <Var>rqsac_ave</Var>
        <Expr>
          <ValueExpr>
            <ListValueExpr>
              <Value>0</Value>
            </ListValueExpr>
          </ValueExpr>
        </Expr>
      </VariableDeclr>
      <VariableDeclr>
        <Text></Text>
        <Properties></Properties>
        <Var>nofcalls</Var>
        <Expr>
          <ValueExpr>
            <ListValueExpr>
              <Value>0</Value>
            </ListValueExpr>
          </ValueExpr>
        </Expr>
      </VariableDeclr>

      <AgreementTerm>
        <Text></Text>
        <Properties></Properties>
        <ID>term4</ID>
        <Precondition>
          <CountExpr>
            <FuncExpr>
              <Text></Text>
              <Properties></Properties>
              <FuncOp>
                <ArithmeticOp>
                  <Divide>
                    <ValueExpr1>
                      <FuncExpr>
                        <Text></Text>
                        <Properties></Properties>
      <FuncOp>
        <ArithmeticOp>
          <Add>
            <ValueExpr1>
              <FuncExpr>
                <Text></Text>
                <Properties></Properties>
                <FuncOp>
                  <ArithmeticOp>
                    <Multiply>
                      <ValueExpr1>
                        <ID>rqsac_ave</ID>
                      </ValueExpr1>
                      <ValueExpr2>
                        <ID>nofcalls</ID>
                      </ValueExpr2>
                    </Multiply>
                  </ArithmeticOp>
                </FuncOp>
              </FuncExpr>
            </ValueExpr1>
            <ValueExpr2>
              <FuncExpr>
                <Text></Text>
                <Properties></Properties>
                <FuncOp>
                  <ArithmeticOp>
                    <Substract>
                      <ValueExpr1>
                        <ID>rqstcv</ID>
                      </ValueExpr1>
                      <ValueExpr2>
                        <CONST>
                          <Value>1</Value>
                        </CONST>
                      </ValueExpr2>
                    </Substract>
                  </ArithmeticOp>
                </FuncOp>
              </FuncExpr>
            </ValueExpr2>
          </Add>
        </ArithmeticOp>
      </FuncOp>
    </FuncExpr>
  </ValueExpr1>
  <ValueExpr2>
    <FuncExpr>
      <Text></Text>
      <Properties></Properties>
      <FuncOp>
        <ArithmeticOp>
          <Add>
            <ValueExpr1>
              <ID>nofcalls</ID>
            </ValueExpr1>
            <ValueExpr2>
              <CONST>
                <Value>1</Value>
              </CONST>
            </ValueExpr2>
          </Add>
        </ArithmeticOp>
      </FuncOp>
    </FuncExpr>
  </ValueExpr2>
</Divide>
```

```
                    </ArithmeticOp>
                  </FuncOp>
                </FuncExpr>
              </CountExpr>
            </Precondition>
            <Guaranteed>
              <Text></Text>
              <Properties></Properties>
              <State>
                <ID>gstate4</ID>
                <Constraint>
                  <FuncExpr>
                    <Text></Text>
                    <Properties></Properties>
                    <Operator>http://www.slaatsoi.org/coremodel#less_than</Operator>
                    <Parameter>
                      <ID>rqsac_ave</ID>
                    </Parameter>
                    <Parameter>
                      <CONST>
                        <Value>30</Value>
                      </CONST>
                    </Parameter>
                  </FuncExpr>
                </Constraint>
              </State>
            </Guaranteed>
          </AgreementTerm>
        </Assertion>
    </Anomalies>
  </AssessmentScheme>

  <ValidityTests/>
  <MonitoringConfigurations>
    <MonitoringConfiguration Id="Id4">
      <Component type="REASONER">
        <EndPoint>EndPoint0</EndPoint>
      </Component>
      <Component type="REASONER">
        <EndPoint>EndPoint1</EndPoint>
      </Component>
      <ConcreteProperty>
        <formula formulaId="formulaId0" type="type2" forChecking="true" diagnosisRequired="false"
threatDetectionRequired="false">
          <quantification>
            <quantifier>forall</quantifier>
            <timeVariable>
              <varName>varName0</varName>
              <varType>TimeVariable</varType>
            </timeVariable>
          </quantification>
          <body>
            <predicate negated="false" unconstrained="false" recordable="false" abducible="false">
              <initially>
                <fluent name="name0">
                  <variable persistent="false" forMatching="true">
                    <varName>varName2</varName>
                    <varType>varType0</varType>
                    <value>value0</value>
                  </variable>
                  <variable persistent="false" forMatching="true">
                    <varName>varName3</varName>
                    <varType>varType1</varType>
                    <value>value1</value>
                  </variable>
                </fluent>
                <timeVar>
            <varName>varName4</varName>
            <varType>TimeVariable</varType>
         </timeVar>
      </initially>
   </predicate>
   <operator>and</operator>
   <predicate negated="false" unconstrained="false" recordable="false" abducible="false">
      <initially>
         <fluent name="name1">
            <variable persistent="false" forMatching="true">
               <varName>varName5</varName>
               <array>
                  <type>type4</type>
               </array>
            </variable>
            <variable persistent="false" forMatching="true">
               <varName>varName6</varName>
               <array>
                  <type>type5</type>
               </array>
            </variable>
         </fluent>
         <timeVar>
            <varName>varName7</varName>
            <varType>TimeVariable</varType>
         </timeVar>
      </initially>
   </predicate>
   <operator>and</operator>
   <relationalPredicate>
      <lessThanEqualTo>
         <operand1>
            <constant>
               <name>name2</name>
               <value>value2</value>
            </constant>
         </operand1>
         <operand2>
            <operationCall>
               <name>name3</name>
            </operationCall>
         </operand2>
      </lessThanEqualTo>
      <timeVar>
         <varName>varName8</varName>
         <varType>TimeVariable</varType>
      </timeVar>
   </relationalPredicate>
</body>
<head>
   <relationalPredicate>
      <notEqualTo>
         <operand1>
            <operationCall>
               <name>name4</name>
            </operationCall>
         </operand1>
         <operand2>
            <expresion persistent="false" forMatching="true">
               <varName>varName9</varName>
               <varType>varType2</varType>
               <value>value3</value>
            </expresion>
         </operand2>
      </notEqualTo>
      <timeVar>
         <varName>varName10</varName>
```

```
              <varType>TimeVariable</varType>
            </timeVar>
          </relationalPredicate>
          <operator>and</operator>
          <relationalPredicate>
            <notEqualTo>
              <operand1>
                <operationCall>
                  <name>name5</name>
                </operationCall>
              </operand1>
              <operand2>
                <operationCall>
                  <name>name6</name>
                </operationCall>
              </operand2>
            </notEqualTo>
            <timeVar>
              <varName>varName11</varName>
              <varType>TimeVariable</varType>
            </timeVar>
          </relationalPredicate>
          <operator>and</operator>
          <timePredicate>
            <timeLessThan>
              <timeVar1>
                <time>
                  <varName>varName12</varName>
                  <varType>TimeVariable</varType>
                </time>
                <minus>0</minus>
                <plus>0</plus>
              </timeVar1>
              <timeVar2>
                <time>
                  <varName>varName13</varName>
                  <varType>TimeVariable</varType>
                </time>
                <plus>0</plus>
                <minus>0</minus>
              </timeVar2>
            </timeLessThan>
          </timePredicate>
        </head>
      </formula>
    </ConcreteProperty>
  </MonitoringConfiguration>
</MonitoringConfigurations>

<EvidenceAggregation>
  <StartDate>"2013-01-01"</StartDate>
  <Intervals intervalsTime="720" intervalUnit="hours"/>
  <FunctionalAggregatorId>Boolean</FunctionalAggregatorId>
  <IntermediateResults>True</IntermediateResults>
</EvidenceAggregation>

<LifeCycleModel>
  <states>
    <state>
      <atomicState stateId="state1" name="Activated" description="Initial State"/>
    </state>
    <state>
      <compositeState stateId="compstate1" name="ContinuousMonitoring">
        <substate>
          <states>
            <state>
              <compositeState stateId="compstate2" name="Issuing">
      <substate>
         <states>
            <state>
               <atomicState stateId="state2" name="Pre-Issued"/>
            </state>
            <state>
               <atomicState stateId="state3" name="Issued"/>
            </state>
         </states>
         <transitions>
            <transition From="state2" To="state3">
               <WhenCondition negated="true">
                  <LogicalExpression negated="true">
                     <EvidenceAggregation/>
                  </LogicalExpression>
               </WhenCondition>
            </transition>
         </transitions>
      </substate>
   </compositeState>
</state>
<state>
   <compositeState stateId="compstate3" name="Anomaly-Audit">
      <substate>
         <states>
            <state>
               <atomicState stateId="state4" name="AnomalySelection"/>
            </state>
            <state>
               <atomicState stateId="state5" name="AnomalyInspection"/>
            </state>
         </states>
         <transitions>
            <transition From="state4" To="state5">
               <GuardCondition negated="true">
                  <LogicalExpression negated="true">
                     <WhenAnomalySelected/>
                  </LogicalExpression>
               </GuardCondition>
            </transition>
            <transition From="state5" To="hstate1">
               <GuardCondition negated="true">
                  <LogicalExpression>
                     <WhenAnomalyResolved/>
                  </LogicalExpression>
               </GuardCondition>
            </transition>
            <transition From="state5" To="state4">
               <GuardCondition negated="false">
                  <LogicalExpression>
                     <UnresolvedAnoly/>
                  </LogicalExpression>
               </GuardCondition>
            </transition>
            <transition From="state5" To="state8">
               <GuardCondition>
                  <LogicalExpression>
                     <WhenUnresolvedAnomaly/>
                  </LogicalExpression>
               </GuardCondition>
            </transition>
         </transitions>
      </substate>
   </compositeState>
</state>
<state>
   <compositeState stateId="compstate4">
```

```
            <substate>
               <states>
                  <state>
                     <atomicState stateId="state6" name="ConflictSelection"/>
                  </state>
                  <state>
                     <atomicState stateId="state7" name="ConflictInspection"/>
                  </state>
               </states>
               <transitions>
                  <transition From="state6" To="state7">
                     <GuardCondition negated="true">
                        <LogicalExpression>
                           <WhenConflictSelected/>
                        </LogicalExpression>
                     </GuardCondition>
                  </transition>
                  <transition From="state7" To="hstate1">
                     <GuardCondition negated="true">
                        <WhenConflictResolved/>
                     </GuardCondition>
                  </transition>
                  <transition From="state7" To="state6">
                     <GuardCondition negated="false">
                        <LogicalExpression>
                           <UnresolvedConflict/>
                        </LogicalExpression>
                     </GuardCondition>
                  </transition>
                  <transition From="state7" To="state8">
                     <GuardCondition>
                        <LogicalExpression>
                           <WhenUnresolvedConflict/>
                        </LogicalExpression>
                     </GuardCondition>
                  </transition>
               </transitions>
            </substate>
         </compositeState>
      </state>
   </states>
   <transitions>
      <transition From="compstate2" To="state4">
         <WhenCondition>
            <Condition>
               <Anomaly>
                  <Anomalies Id="2101"/>
               </Anomaly>
            </Condition>
         </WhenCondition>
      </transition>
      <transition From="compstate2" To="state6">
         <WhenCondition>
            <Condition>
               <conflictCondition>
                  <Conflict Id="1100"/>
               </conflictCondition>
            </Condition>
         </WhenCondition>
      </transition>
      <transition From="compstate2" To="state1">
         <WhenCondition>
            <Condition>
               <expirationCondition>
                  <ExpirationCondition Id="987"/>
               </expirationCondition>
            </Condition>
```

```
                </WhenCondition>
              </transition><transition From="compstate2" To="state4">
                <WhenCondition>
                  <Condition>
                    <expirationCondition>
                      <ExpirationCondition Id="987"/>
                    </expirationCondition>
                  </Condition>
                  <Condition>
                    <Anomaly>
                      <Anomalies Id="2101"/>
                    </Anomaly>
                  </Condition>
                </WhenCondition></transition>
            </transitions>
          </substate>
        </compositeState>
      </state>
      <state>
        <atomicState stateId="state8" name="Revoked"/>
      </state>
    </states>
    <historyState stateId="hstate1" refersToStateId="compstate2"/>
    <transitions>
      <transition From="state1" To="state2">
        <WhenCondition>
          <Condition>
            <evidenceSufficiencyCondition>
              <EvidenceSufficiencyCondition Id="1011"/>
            </evidenceSufficiencyCondition>
          </Condition>
        </WhenCondition>
      </transition>
    </transitions>
  </LifeCycleModel>

</model:CertificationModel>
```

10.5. TC Support for Certification Schema (.xsd file)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="unqualified" blockDefault="substitution"
    elementFormDefault="unqualified" targetNamespace="urn:cumulus:tcsupport"
    xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tc="urn:cumulus:tcsupport"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

<import namespace="http://www.w3.org/2000/09/xmldsig#"
        schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd" />

  <element name="TCSupport" type="tc:TCSupportType" />

  <complexType name="TCSupportType">
    <sequence>
      <element name="PlatformState" type="tc:PlatformStateType" />
      <element name="ApplicationState" type="tc:ApplicationStateType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="StateBoundKey" type="tc:StateBoundKeyType" />
    </sequence>
    <attribute name="TPMVersion" type="string" use="required" fixed="1.2"/>
  </complexType>

  <complexType name="PlatformStateType">
    <sequence>
      <element name="Hash" type="tc:DigestValueType"/>
      <element name="PCRNumber" type="unsignedLong" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="ApplicationStateType">
    <sequence>
      <element name="Hash" type="tc:DigestValueType"/>
      <element name="PCRNumber" type="unsignedLong"/>
      <element name="ApplicationRef" type="tc:ApplicationRefType"/>
    </sequence>
    <attribute name="IntegrityMethod" type="anyURI" use="required"/>
  </complexType>

  <complexType name="ApplicationRefType">
    <sequence>
      <element name="ElementRef" type="anyURI" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <complexType name="StateBoundKeyType">
    <sequence>
      <element ref="ds:KeyInfo"/>
    </sequence>
    <attribute name="TPMKeyType" type="tc:TPMKeyTypeType" use="required"/>
    <attribute name="BoundTo" type="tc:BoundToType" use="required"/>
  </complexType>

  <simpleType name="TPMKeyTypeType">
    <restriction base="string">
      <enumeration value="Non-MigratableKey" />
      <enumeration value=" MigratableKey" />
    </restriction>
  </simpleType>

  <simpleType name="BoundToType">
    <restriction base="string">
      <enumeration value="PlatformState" />
      <enumeration value="PlatformAndApplicationState" />
    </restriction>
  </simpleType>

  <complexType name="DigestValueType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute name="AlgRef" type="anyURI" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
</schema>
```

10.6. Specification of the BNF grammar of SecureSLA*

```
//--- SLA Template ---
assertion : 'assertion { agreedAt = '( time | 'n/a' ) 'effectiveFrom =' time
'effectiveUntil =' time 'templateId =' id slatemplate '}';
slatemplate : 'sla_template {' slatemplatecontent '}';
slatemplatecontent : 'uuid =' id 'sla_model_version = sla_at_soi_sla_model_v1.0'
(agreementparty)+ (interfacedeclr)+ (variabledecl)* (agreementterm)+;

//--- Agreement Parties ---
agreementparty : 'party { id =' id 'role =' agreementrole (operative)* '}' |
'abstractparty { id = ' id 'role =' agreementrole '}';

//STANDARD TERMS - AGREEMENT ROLES
agreementrole : id;
operative : 'operative { id =' id '}';

//--- Interface Declarations ---
interfacedeclr : 'interfacedecl { id =' id 'providerref =' providerref
(endpoint)* choice '}' ;
choice : interfaceref | interfacespec | resourcetype;
providerref : (id) | 'provider' | 'customer';
```

```
endpoint : 'endpoint { id =' id 'location =' (id | 'customlocation') 'protocol
=' endpointstnd '}';

//STANDARD TERMS - PROTOCOLS
endpointstnd : 'TELEPHONE' | 'SOAP' | 'Email' | 'SMS' | 'REST' | 'XMPP' | 'HTTP'
| 'Post_mail' | 'FAX' | 'SSH';
interfaceref : 'interface_ref (' id ')';
resourcetype : 'resource_type { name =' id '}';

//---Variable Declarations---
variabledecl :
id 'is' (id)* '(' valueexpr ')' ('initially' valueexpr)* (valueexpr)* |
 id 'is' (id)* valueexpr ('initially' valueexpr)* (valueexpr)* | id 'is' '('
(id)* valueexpr ')' ('initially' valueexpr)* (valueexpr)* |  id 'is' (id)* '('
domainexpr (consta)* ')' ('initially' valueexpr)* (valueexpr)* | id 'is' (id)*
'(' domainexpr (consta)* ')' ('and' | 'or')* '(' domainexpr (consta)* ')'
('initially' valueexpr)* (valueexpr)* | list_var_decl | service_variable_decl;


service_variable_decl : id 'is' serviceref;
list_var_decl : id 'is' 'list[ index=' id ',' (valueexpr)* (domainexpr)* (','
valueexpr)* (',' domainexpr)* ']’


//--- Agreement Terms ---
agreementterm  : 'agreement_term { id =' id ('precondition {' constraintexpr
'}')* (variabledecl)* (guarantee)+ '}';
guarantee : (guaranteedstate | guaranteedaction);

//--- Guaranteed States ---
guaranteedstate : 'guaranteedstate { id =' id ('priority =' consta)*
('precondition { ' constraintexpr '}')* (constraintexpr)* '}' ;

// --- Guaranteed Actions ---
guaranteedaction : 'guaranteedaction { id =' id 'actor =' id 'policy ='
actionpolicy 'trigger =' (eventexpr | id) actionpostcondition '}';
actionpolicy : actionstnd ;

//STANDARD TERMS - ACTION POLICIES
actionstnd : 'MANDATORY' | 'OPTIONAL' | 'FORBIDDEN';
actionpostcondition : (invocation_action | var_update_action);

//--- Invocation Actions ---
invocation_action : 'invoke { endpoint =' id 'operation =' id (invocationparam)*
'}';
invocationparam : 'param { name =' id 'value =' valueexpr '}';
extensionlist : 'extensionlist {'(id)+ '}';

//--- Variable Update Actions ---

var_update_action:
id 'is' '(' valueexpr ')';

//--- Interface Specifications ---
interfacespec : 'interfacespec { name =' id (extensionlist)* (operation)* '}';
operation : 'operation { name =' id (input)* (output)* (related)* (faultlist)*
'}';
input : 'input {' propertycontent '}';
output : 'output {' propertycontent '}';
related : 'related {' propertycontent '}';
```

```
propertycontent : 'name =' id 'datatype =' id 'domain = (' domainexpr ')'
'auxiliary =' bool;
faultlist : 'faultlist {' (id)* '}';
bool : ('true') | ('false');

//--- Service References ---
serviceref : 'serviceref {' (id)+ '}';

//--- Constraint Expressions ---
constraintexpr : (id)* valueexpr | funcop |
valueexpr domainexpr |
comparisonop ('(')* (valueexpr)* (')')* |
domainexpr valueexpr |
valueexpr domainexpr consta |
'not' constraintexpr |
'(' constraintexpr 'and' constraintexpr ')' |
'(' constraintexpr 'or' constraintexpr ')' |
'(' constraintexpr ')' 'and' '(' constraintexpr ')' |
'(' constraintexpr ')' 'or' '(' constraintexpr ')' | count ;

domainexpr : comparisonop '(' valueexpr ')' |
comparisonop valueexpr |
'not' '(' domainexpr ')' |
'(' domainexpr 'and' domainexpr ')' |
'(' domainexpr 'or' domainexpr ')' |
comparisonop '(' valueexpr ')' |
primitivedomain;

primitivedomain : 'type(integer)' | 'type(float)' | 'type(string)' |
'type(boolean)';

'equals' |'not_equals' | 'less_than' | 'less_than_or_equal'
comparisonop : comparisonstnd;

comparisonstnd : 'identical_to' |
'equals' |'not_equals' | 'less_than' | 'less_than_or_equal' | 'greater_than' |
'greater_than_or_equal' | ' matches' | 'isa';

//--- Event Expressions ---
eventexpr : specialisation | intersection | union | difference | call_event |
periodic | schedule | eventtime | fault | violated | warned | recovered | timeof
| reply_event;

//STANDARD TERMS - EVENTS
//they will need to be defined as a restriction for the sub-element //Operator
in EventExprType in the XML schema

specialisation : 'specialisation [' eventexpr ',' constraintexpr ']' |
'specialisation [' valueexpr ',' constraintexpr ']';
intersection : 'intersection [' eventexpr ',' eventexpr ']' | 'intersection ['
eventexpr ',' eventexprvar ']' | 'intersection [' eventexpr ',' eventexprvar ']'
| 'intersection [' valueexpr ',' valueexpr ']' | 'intersection [' valueexpr ','
eventexprvar ']' | 'intersection [' eventexpr ',' valueexpr ']';
union : 'union [' eventexpr ',' eventexpr ']' | 'union [' eventexpr ','
eventexprvar ']' | 'union [' valueexpr ',' valueexpr ']' | 'union [' valueexpr
',' eventexprvar ']' | 'union [' eventexpr ',' valueexpr ']';
difference : 'difference [' eventexpr ',' eventexpr ']' | 'difference ['
eventexpr ',' eventexprvar ']' | 'difference [' valueexpr ',' valueexpr ']' |
'difference [' valueexpr ',' eventexprvar ']' | 'difference [' eventexpr ','
valueexpr ']';
call_event : 'invocation [' (inv_ref)* ']';
```

```
inv_ref : service_variable_decl | invocation_action ;
periodic : 'periodic [' NUMBER duration ']' | 'periodic [' id ']'
(invocation_action)* (',')* (invocation_action)*;
schedule : 'schedule [' time (time)* ']';
eventtime : 'time [' NUMBER '-' NUMBER '-' NUMBER 'T' NUMBER ':' NUMBER 'UTC'
']';
fault : 'fault [' service_variable_decl ']';
violated : 'violated [' constraintexpr ']';
warned : 'warned [' constraintexpr ',' warned_ratio ']';
warned_ratio : NUMBER'.'NUMBER;
recovered : 'recovered [' constraintexpr ']';
reply_event : 'reply [' (inv_ref)* ']';
timeof : 'timeof [' eventexpr ']' | 'timeof [' valueexpr ']';

eventexprvar : id 'is' (id)* eventexpr;

//--- Functional Expressions ---
funcexpr : funcop (',')*
(id | bool | consta | time | serviceref | eventexpr)* (consta)* | funcop (',')*
'(' (id | bool | consta | time | serviceref | eventexpr)* ')' (consta)* |
funcop (',')* '(' (id | bool | consta | time | serviceref | eventexpr)* ','
(',')* (id | bool | consta | time | serviceref | eventexpr)* ')' (consta)* |
funcop (',')* '(' (funcexpr ',')* funcexpr ')' (consta)* | funcop (',')* '('
funcexpr ',' (id | bool | consta | time | serviceref | eventexpr)* ')' (consta)*
| funcop (',')* '(' (id | bool | consta | time | serviceref | eventexpr)* ','
funcexpr ')' (consta)* ;
funcop : id | arithmeticop | contextop | qosterm | setop | timeseriesop |
listop;
setop : sum | std | mean | median | modeoff | max | min;

listop : inserttolist | removefromlist | updateinlist;
inserttolist: 'insert(' (id)* (list_var_decl)*',' constraintexpr ',' valueexpr
')';
removefromlist: 'remove(' (id)* (list_var_decl)*',' constraintexpr ',' valueexpr
')';
updateinlist: 'update(' (id)* (list_var_decl)*',' constraintexpr ',' valueexpr
')';


//STANDARD TERMS - ARITHMETIC FUNCTIONS

add : ('add') '(' arithmeticop ',' arithmeticop ')' | ('add') '(' valueexpr ','
arithmeticop ')' | ('add') '(' arithmeticop ',' valueexpr ')' | ('add') '('
valueexpr ',' valueexpr ')';
substract : ('substract') '(' arithmeticop ',' arithmeticop ')' | ('substract')
'(' valueexpr ',' arithmeticop ')' | ('substract') '(' arithmeticop ','
valueexpr ')' | ('substract') '(' valueexpr ',' valueexpr ')';
multiply : ('multiply') '(' arithmeticop ',' arithmeticop ')' | ('multiply') '('
valueexpr ',' arithmeticop ')' | ('multiply') '(' arithmeticop ',' valueexpr ')'
| ('multiply') '(' valueexpr ',' valueexpr ')';
divide : ('divide') '(' arithmeticop ',' arithmeticop')' | ('divide') '('
valueexpr ',' arithmeticop ')' | ('divide') '(' arithmeticop ',' valueexpr ')' |
('divide') '(' valueexpr ',' valueexpr ')';
modulo : ('modulo') '(' arithmeticop ',' arithmeticop ')' | ('modulo') '('
valueexpr ',' arithmeticop ')' | ('modulo') '(' arithmeticop ',' valueexpr ')' |
('modulo') '(' valueexpr ',' valueexpr ')';
round : ('round') '(' arithmeticop ',' arithmeticop ')' | ('round') '('
valueexpr ',' arithmeticop ')' | ('round') '(' arithmeticop ',' valueexpr ')' |
('round') '(' valueexpr ',' valueexpr ')';
```

```
contextop : contextstnd;
//STANDARD TERMS - CONTEXT FUNCTIONS
contextstnd : 'time_is' time | 'day_is' day | 'month_is' month | 'year_is'
NUMBER;
day : 'MONDAY' | 'TUESDAY' | 'WEDNESDAY' | 'THURSDAY' | 'FRIDAY' | 'SATURDAY' |
'SUNDAY';
month : 'JANUARY' | 'FEBRUARY' | 'MARCH' | 'APRIL' | 'MAY' | 'JUNE' | 'JULY' |
'AUGUST' | 'SEPTEMBER' | 'OCTOBER' | 'NOVEMBER' | 'DECEMBER' ;

//STANDARD TERMS - QOS TERMS
qosterm : qos_availability | qos_accessibility | qos_arrivalrate |
qos_datavolume | qos_throughput | qos_completiontime | qos_mttr | qos_mttf |
qos_mttv | qos_reliability | qos_isolation | qos_accuracy | qos_nonrepudiation |
qos_supportedstandards | qos_regulatory | qos_integrity | qos_authentication |
qos_auditability | qos_authorisation | qos_data_encryption;
qos_availability : 'availability (' service_variable_decl ')';
qos_accessibility : 'qos_accessibility (' service_variable_decl ')';
qos_arrivalrate : 'qos_arrivalrate (' service_variable_decl ')';
qos_datavolume : 'qos_datavolume (' service_variable_decl ')';
qos_throughput : 'qos_throughput (' service_variable_decl ')';
qos_completiontime : 'qos_completiontime (' service_variable_decl ')';
qos_mttr : 'qos_mttr (' service_variable_decl ')';
qos_mttf : 'qos_mttf (' service_variable_decl ')';
qos_mttv : 'qos_mttv (' service_variable_decl ')';
qos_reliability : 'qos_reliability (' service_variable_decl ')';
qos_isolation : 'qos_isolation (' service_variable_decl ')';
qos_accuracy : 'qos_accuracy (' service_variable_decl ')';
qos_nonrepudiation : 'qos_nonrepudiation (' service_variable_decl ')';
qos_supportedstandards : 'qos_supportedstandards (' service_variable_decl ')';
qos_regulatory : 'qos_regulatory (' service_variable_decl ')';
qos_integrity : 'qos_integrity (' service_variable_decl ')';
qos_authentication : 'qos_authentication (' service_variable_decl ')';
qos_auditability : 'qos_auditability (' service_variable_decl ')';
qos_authorisation : 'qos_authorisation (' service_variable_decl ')';
qos_data_encryption : 'qos_data_encryption (' service_variable_decl ')';

//STANDARD TERMS - AGGREGATE (SET) FUNCTIONS
sum : 'sum (' seriesvar ')';
std : 'std (' seriesvar ')';
mean : 'mean (' seriesvar ')';
median : 'median (' seriesvar ')';
modeoff : 'modeoff (' seriesvar ')';
max : 'max (' seriesvar ')';
min : 'min (' seriesvar ')';
seriesvar : id 'is' series ;
series : 'series (' (funcexpr ',')* eventexpr (',' count)* ')';
count : 'count (' (funcexpr ',')* eventexpr (domainexpr)* ')';


//STANDARD TERMS - TIME SERIES
timeseriesop : series | value;
value : 'series (' funcexpr ',' eventexpr ')';

//---Value Expressions---
valueexpr : id | consta | bool | time | serviceref | eventexpr | funcexpr;
expr : (constraintexpr) | (domainexpr) | (valueexpr);
consta : (',')* (NUMBER)+'.'(NUMBER)* metric;
id : PATH | UUID;
PATH : [a-z]+ ('::' [a-z]+)* ('::' NUMBER)*;
UUID : ([a-z]+) | ([a-z]+ [0-9]+) ;
WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

```
NUMBER : [0-9]+ ;
time : NUMBER ':' NUMBER ':' NUMBER ':' NUMBER;

//STANDARD TERMS - METRIC UNITS
metric : area | datarate | datasize | energy | length | frequency  | ratio |
power | txrate | weight | currency  | duration | unit;
area : 'mm2' | 'um2';
datarate : 'b_per_s' | 'Kb_per_s' | 'Mb_per_s';
datasize : 'bit' | 'byte' | 'KB' | 'MB' | 'GB';
energy : 'J' | 'KJ' | 'Wh' | 'KWh' | 'mWh';
length : 'm' | 'cm' | 'mm';
duration : 's' | 'tick' | 'ms' | 'us' | 'minutes' | 'hrs' | 'day' | 'week' |
'month' | 'year';
frequency : 'hz' | 'KHz' | 'MHz' | 'GHZ' | 'rpm';
ratio : 'percentage';
power : 'W' | 'mW' | 'kW';
txrate : 'tx_per_s' | 'tx_per_m' | 'tx_per_h';
weight : 'g' | 'mg' | 'kg';
currency : 'EUR' | 'USD';
unit : ‘units’;
```