Security Certification of Modern Distributed Systems

Marco Anisetti, Claudio A. Ardagna, Ernesto Damiani, Nabil El Ioini

June 21, 2023

Preface

Preface

Chapter 1

Introduction

Modern distributed systems are at the intersection of many key technologies and IT evolutions, from cloud to IoT, from microservices architecture to devops, from wired to mobile networks, from big data to machine learning and artificial intelligence. Current implementations are permeating our society bringing distributed systems at the core of (safety-critical) services and applications and, at the same time, making people yet another components of distributed systems themselves.

This complex environment and the unpredictability that is introduced when a person is integrated within the system execution flow, introduce the need of rethinking existing security approaches, pushing towards security assurance. According to [**?**], assurance can be defined as the way to gain justifiable confidence that infrastructure and/or applications will consistently demonstrate one or more non-functional (e.g., security) properties, and operationally behave as expected despite failures, malfunctioning or attacks. Assurance is a wider notion than includes methodologies for collecting and validating evidence supporting specific non-functional properties.

It includes audit, compliance and certification techniques, which has been recently applied in many domains of our society, with specific reference to the digital society. In this book we focus on certification as a primary means of assurance and discuss its evolution across IT evolutions.

1.1 Context and motivation

Distribute systems plays a major role in every domain of our society, and highly affect the distribution of services and applications to final users. Their widespread is permeating all aspects of our lives; however, the high quality standards of these systems do not find the counterpart in the security and privacy domains. The same problems that we witnessed with the advent of software systems and analyzed in our seminal book on software certification [**?**] are even more critical for distributed systems. Many users, as well as businesses and

vendors, still express little trust in the correctness and reliability of the systems and services they use every day.

The book in [**?**] analyzed the problem of software system certification from a wide perspective, considering issues and requirements when static and very complex systems (e.g., a Linux operating system) underwent a complete certification software. The book presented different certification scheme then focusing on Common Criteria, the de-facto standard for software system certification. This book departs from the analysis in [**?**] and presents the evolution of security certification across IT evolution, towards the certification of modern distributed systems. In the last decade, industries, as well as governments, has been at the forefront of the effort pushing towards the definition of new certification schemes that address the peculiarities of modern distributed systems. For instance, ENISA has been appointed by the European Commission as the responsible for developing EU cybersecurity certification which provides evidence of compliance to a given level of trust. The mission of ENISA in the area of the EU cybersecurity certification framework is "*to proactively contribute to the emerging EU framework for the ICT certification of products and services and to carry out the drawing up of candidate certification schemes in line with the Cybersecurity Act, and additional services and tasks*".

In this book, we consider the huge research effort devoted to find methods for checking distributed system quality that may result in sound security, safety and dependability guarantees. This is mandatory due to the increasing pervasiveness and involvement of people in the system working. Specifically, researchers are addressing the following questions:

- What can we certify today? How can we address the peculiarities of distributed systems?
- How can we trust a certificate to be accurate across distributed systems changes?
- How can we measure our trust on dynamic and evolving systems?
- What can be done at design time to integrate development and certification processes?
- How can we address composite systems?

We focus on the certification of security-related properties, that is, the process proving a number of high level security properties by collecting a sufficient amount of evidence. Traditionally, security certification has been mostly applied when customers need to have a quantitative measure of the security level of their IT products, since it provides the possibility to choose systems that best match their security needs. A wide range of certified IT products exist, including operating systems, firewalls, and smart cards. However, among those certified products just a few are modern systems based on (micro-)services. In addition, among existing certification schemes just a few are applicable to modern systems that: *i)* are complex and composite, *ii)* subject to quick versioning and adaptations, *iii)* mix heterogeneous technologies.

The discussion in this book is driven by three main requirements that are summarized below:

multi-dimensional certification. Certification is not only a matter of software artifacts. Certification concerns many aspects of system life cycle, from the development process to the verification process and even the deployment process. A certification scheme should consider evidence collected in multiple dimensions, contributing to build a more correct and comprehensive view of the certified system.

Composite certification. Modern systems are implemented as a smart composition of services and components. Certifying a system as a whole introduces inaccuracy in the certification results. This introduces the need of certification schemes that are capable of inferring the properties of the whole system starting from the properties of its components.

Continuous certification. Modern systems are modified and deployed at fast rates with dozen of service versions released every day. Modern certification schemes must reduce the monetary costs and the time needed to certify a specific system, supporting quick and partial re-certification across system changes. Smart certificate life cycle management approach must be defined, increasing certificate support over time.

Chain of trust. Traditional certification assumes a trusted third party (i.e., certification authority) that is responsible for the entire management of the certification process. This assumption is not sound when modern systems are the target of certification. Modern systems in fact are continuously developed and deployed impeding the availability of a certification authority for the entire certification process, which last for the entire system life cycle. New chain of trust routed at a trust third party are needed to support sound certification schemes.

Deterministic vs non-deterministic. Modern systems are quickly moving from deterministic to non-deterministic behavior, where their behavior is driven by machine learning and artificial intelligence. This evolution is hampering existing schemes asking for new approach to certification that should start with a new definition of properties.

1.2 Structure of the book

TBC

Chapter 2

Part 1: History of Software Security Certification

Today, computer systems play a central role in all aspects of modern life. We do not only depend on them for secondary tasks, rather, most of the highly critical infrastructures such as health care, transportation, and telecommunication depend on them. However, in spite of this pervasive presence of computer systems in our daily activities, many users feel reluctant in trusting them. In the majority of instances, this can be primarily attributed to the fact that these systems lack a proper evaluation process that guarantees their reliability and trustworthiness. In this chapter we start by defining what software security certification is and, then we look at some of the leading evaluation and certification schemes that have been adopted by governments and the industry over the years.

2.1 Software Security Certification

Software security certification refers to the process of verifying that a software product meets a set of predefined security requirements, following an evidence-based process. This can involve different aspects including, system architecture design, code review, and testing in order to ensure that the software system can be awarded the certification stamp.

A variety of certification schemes have been developed over the years, each with specific scope, requirements, and standards. Some certification schemes focus on the technical aspects of the software, such as its performance, security, and reliability, others instead focus on the development process and organizational aspects such as usage policies and the operational environment.

To certify the security of a software system, dedicated trusted organizations oversee the evaluation process by following well-documented processes and protocols [1]. The main task of these organizations is to review software systems and conduct various evaluation procedures to ensure that systems meet the necessary standards. If the software passes all the evaluation milestones, it is

granted a certificate, which is used to prove to potential customers that the software has been independently reviewed and deemed to be of high quality. Obtaining software certification is beneficial for companies as it increases confidence in their products and therefore, the credibility of the company. At the same time, the certificate provides assurance to customers regarding the reliability of their systems and allows them to hold their vendors accountable in case of failures or behaviours that invalidate the issued certificate.

2.2 History of Software Security Certification

The history of software security certification can be traced back to the beginning of the Internet. The issues that were brought by the Internet in terms of access/data control and privacy have challenged the potential success of an open-world wild network capable of providing easy access to resources while maintaining a high level of security and privacy [2]. Prior to this, in the early days of personal computing, there was little need for such certifications as the ability to seamlessly transfer data from one machine to another was nonexistent. Users were forced to manually move data to the intended recipient. Even though policies and regulations were put in place to regulate who has access to and how data is managed, they were at the management level rather than at the software level. The emergence of the Internet marked a significant turning point. By connecting standalone machines, the transfer of information became faster and more convenient. The Internet bridged the gap, allowing users to rely on software systems to manipulate and transmit data without the need for manual actions and reducing transaction costs associated with information exchange.

This new environment helped define the vision for software market expansion, the software industry has expanded exponentially since then, with companies targeting different domains with a specific focus on the enterprise and military segments due to their high financial returns. The increase in commercial solutions and their options eventually comes with a major downside, which is the elevated risk of provisioning insecure software systems. Governments and agencies, in particular, have taken notice of this issue, due to the high impact these systems might have on their operations. As a response, several government bodies have introduced dedicated security certification schemes for the software market. Their aim was to establish rigorous standards and evaluation processes to ensure the reliability, integrity, and safety of software products.

In general, a certification process is driven by the security properties to be certified (e.g., confidentiality, integrity, authenticity), and 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 (i.e., *accredited Lab*) that carries out the property evaluation.

2.2.1 Certification Schemes

The earliest scheme of software security certification can be traced back to the 1970s, when the U.S. Department of Defense (DoD) established the Trusted Computer System Evaluation Criteria (TCSEC) [3]. TCSEC focused primarily on evaluating the security of computer and software systems by introducing a set of security classes. The TCSEC was later revised and became known as the Orange Book, which was a widely-used reference for software security certification. In the 1990s, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) developed the Common Criteria, which is a set of security standards that are still used today for software security certification [1]. The Common Criteria provide a framework for evaluating the security of software and other types of information technology products. In addition to the Common Criteria, there are several other trust models that are commonly used for software security certification, including the Capability Maturity Model (CMM), the Payment Card Industry Data Security Standard (PCI DSS), and the Health Insurance Portability and Accountability Act (HIPAA).

In the subsequent sections, we will provide more details regarding each of the certification schemes aforementioned looking more specifically at their strengths and weaknesses in order to identify the gaps that need to be addressed by future schemes schemes.

2.2.2 Trusted Computer System Evaluation Criteria (TCSEC)

The Trusted Computer System Evaluation Criteria (TCSEC), or Orange Book was the first attempt to come up with a clear and systematic certification process. It has been developed and adopted by the US government for computer systems procurement. TCSEC establishes a complete evaluation framework for accessing the security features of software systems. The primary objective of TCSEC is to lay the foundations for a common language capable of capturing clearly all the concepts and features subject to the certification process. TCSEC targets four Security Levels (Table 2.1) and focuses on seven main security features (Table 2.2).

Table 2.1: TCSEC Security Levels

Security Level	**Description**	**Examples**
Level D - Minimal Protection	This level includes systems that lack formal security controls. this level deals with systems that rely on the security of their operational environment without having secure implementation themselves	Standalone machines with no connection to the network in a secure environment.
Level C - Discretionary Security Protection	These are systems that implement a basic level of security, namely, discretionary access control (DAC), however, the system still relies on the admins to enforce security policies	A typical example is an operating system with user file permissions (e.g., Windows 98)
Level B - Mandatory Protection	This level we start seeing the distinction between admin and regular users roles. Typically, systems implement mandatory access controls (MAC) for users and files which are used to grant access permissions.	Linux operating system that enforces MAC access to system files.
Level A1 - Verified Protection	At this level software systems undergo formal security verification to show their resistance to complex attacks. This level features rigorous standardisation and auditing processes.	A cryptographic module, such as a hardware security module (HSM),
Level A2 - Structured Protection	This level focuses on structured and systematic security requirement documentation and security policies	An operating system with strong cryptographic and key management controls
Level B1 - Labeled Security Protection	This level extends level B by augmenting the security model with additional labels. The labels define the sensitivity and integrity level associated with each subject and object.	VAX/VMS operating system has been certified with level B1.
Level B2 - Structured Protection with Security Policy	At the level, a model of the security policy is required. This allows not only the software product but also its policy to be formally verified	The SEVMS (Secure Enclave Virtual Memory System) has been certified with B2 level.

TCSEC has contributed to creating the first systematic process to address computer security, however, despite its merits, it exhibited several limitations. Given its static nature, TCSEC failed to adapt and support the rapid evolution and complexity of modern systems. Additionally, the emergence of new classes of threats called for up-to-date evaluation criteria.

2.2.3 Information Technology Security Evaluation Criteria (ITSEC)

Information Technology Security Evaluation Criteria (ITSEC) is the former European standard for computer security certification that was developed to assess the security of IT products. It was developed by the European Union in the 1990s and was in use until the early 2000s, when it was replaced by the Common Criteria for Information Technology Security Evaluation (CC) [4].

ITSEC was developed to certify IT products used mainly by EU government agencies and large organizations with critical security requirements. ITSEC was divided into three parts:

- Security Target (ST): a document defining the security requirements for a particular IT product.
- Evaluation Assurance Level: a measure of the level of assurance that the IT product or system met the requirements specified in the ST.
- Security Functional Level: a measure of the level of security functionality provided by the IT product or system.

Seven assurance levels were defined under the ITSEC scheme, namely:

1. E0: This level represents no assurance.
2. E1: Functionally tested.
3. E2: Structurally tested.
4. E3: Methodically tested and checked.
5. E4: Methodically designed, tested, and reviewed.
6. E5: Semi-formally designed and tested.
7. E6: Formally designed and tested.

2.2.4 Canadian Trusted Computer Product Evaluation Criteria (CTEPEC)

Canada has developed its own certification scheme in response to the race towards establishing rigorous criteria for computer systems security. CTEPEC was developed by the Canadian government's cryptologic agency to address

Table 2.2: TCSEC Security Features

Feature	**Description**
Identification and Authentication	identification of all the users and objects in a software system. This could include token, biometric and password-based authentications.
Access Control	These mechanisms are meant to regulate access control to all systems resources. They include role-based access control (RBAC), access control lists (ACLs), and mandatory access control (MAC).
Auditing	the ability of the system to maintain tamper-proof security records that can be used to audit the system behaviour.
Trusted Recovery	In case of security breaches, the system is able to recover while maintaining integrity and its security configuration.
Trusted Communication	provide secure communication channels between all resources.
Object Reuse Protection	The ability to clean and flush any secure data before resources can be reused by other users or entities.
Accountability	any activity or action in the system can be associated with a specific entity or user.

Canadian particular needs [5]. Table 2.3 the six security levels considered by the CTEPEC scheme.

The certification process is carried out by the Canadian Common Criteria Evaluation and Certification Scheme (CCS) laboratories. The process starts by contacting an accredited lab and submitting all the necessary documentation depending on the level the client is seeking to certify. After a pre-evaluation phase, where all the submitted documentation is reviewed, an evaluation planning phase starts, where the lab outlines the concrete evaluation activities to carry out (e.g., verification methodology, timeline, resource allocation). Next, the evaluation phase starts. This phase executes all the pre-defined activities including testing the product's security features, source code review, vulnerability analysis etc. Once this phase is completed an evaluation report is generated, which is then used to decide whether to award the required certification level or not.

To guarantee the validity of the certificate for future releases of the product, a continuous certification maintenance phase has been put in place. This activity requires periodic monitoring of the certified products to ensure their alignment with the specified security requirements of the awarded level.

2.2.5 Common Criteria (CC)

The Common Criteria (CC) for Information Technology Security Evaluation is an international standard for computer security certification [1]. It was first developed in the 1990s as a joint effort between the National Security Agency (NSA) of the United States, the Communications-Electronics Security Group (CESG) of the United Kingdom, and the Federal Office for Information Security (BSI) of Germany. The CC standard is recognized and used by many governments and organizations around the world, and it is often used as a basis for government procurement and regulatory compliance, as well as for private sector security certification programs.

The main goal of the CC standard is to present a uniform and consistent security evaluation process that can be used to assess the security of IT products. Its primary target users are governments, businesses, and large organizations to guarantee with a high level of confidence that the IT products they use meet a predefined level of security.

The CC standard is composed of three parts:

- the Protection Profile (PP): a document describing the security requirements for a particular IT product or system. This represents what security means for the organization seeking the certify the IT product.

- Security Target (ST): a document describing how the IT product meets the requirements specified in the PP.

- Evaluation Assurance Level (EAL): a measure of the level of assurance that the IT product meets the requirements specified in the ST.

Table 2.3: CTEPEC Levels

Level	**Description**
E1	Level E1 apply to software systems with non-sensitive data and low-security requirements. Many commercial off-the-shelf (COTS) software products fall into the category including web browsers and software utilities. Security at this level deals with the possibility of the software system affecting the security of the environment it is hosted in.
E2	Non-critical and low-risk systems can be certified at this level. These systems are checked for their access controls and authentication/authorisation mechanisms.
E3	This level deals with systems that manage sensitive information. To be certified at Level E3, the software system needs to provide evidence that shows stronger security features and stricter controls.
E4	This level adds more restrictions on the operating environment of the software system. It mandates that the operational environment needs to guarantee a high level of security including secure communication, secure hardware components and a secure operating system.
E5	This level deals with highly classified information. Rigorous evaluation processes are used to verify the security features of the products undergoing the certification process at this level.
E6	This is the highest level a software product can achieve under CTCPEC. software systems dealing with highly sensitive data can be certified at this level if they can formally and empirically prove a high level of security for all the considered features.

Over the years, the CC standard has undergone several revisions since it was first introduced. The most recent version, CC v3.1, was released in 2013. It includes a number of updates and improvements over previous versions, including support for new technologies and security challenges including IoT and cloud-based systems.

The CC defines seven Evaluation Assurance Levels (EAL) that represent increasing degrees of assurance.

1. EAL1: Functionally Tested
2. EAL2: Structurally Tested
3. EAL3: Methodically Tested and Checked
4. EAL4: Methodically Designed, Tested, and Reviewed
5. EAL5: Semi-Formally Designed and Tested
6. EAL6: Semi-Formally Verified Design and Tested
7. EAL7: Formally Verified Design and Tested

2.3 Certification Process

Although each scheme has defined its own certification process, they all share the same common pattern. They all rely on an evidence-based certification process of evaluating and certifying products, practices, and systems. The process takes as input the properties and the product to be certified and produces as output a machine-readable certificate, containing the evidence that proves the security level of the required properties. The main advantage of evidence-based certification is its reliance on empirical evidence to support the claims made about a product.

A typical evidence-based certification process involves the following steps:

1. Identify the properties to be certified.
2. Identifying the requirements to be verified for each property.
3. Gathering and reviewing relevant evidence to support those requirements.
4. Evaluating the quality and relevance of the evidence.
5. Determining whether the evidence is sufficient for the required level of security.
6. Granting certification if the evidence meets the necessary criteria.

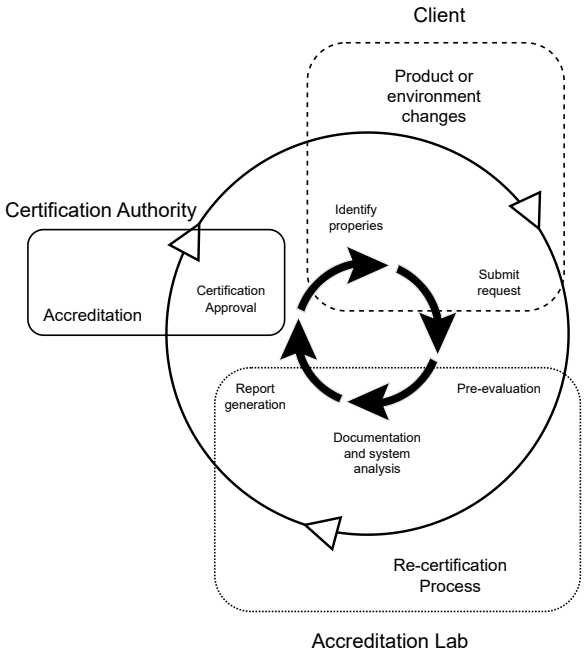

Figure 2.1: Conceptual framework of the certification process.

Based on the conceptual framework in Figure 2.1, the certification process is composed of two cycles and it is carried out by three actors. The inner cycle is initiated by the client who defines the scope of the certification by identifying the features to certify and the required certification level. Upon submitting a certification request, the client needs to support the request with all the essential documentation to facilitate the evaluation process.

The second phase involves the accredited lab, which plans the evaluation process and carries out all the involved activities including product testing, code review and threats analysis. The accredited Lab generates the evidence needed to certify the product on the basis of the provided documentation and the carried-out tests and sends it to the certification authority. If the evidence is sufficient to prove the required property, the certification authority awards a certificate to the client, which includes the certified property, the certification level, and the evidence.

Once the inner cycle is completed it triggers the outer cycle which monitors the certified product and in case there are any changes either in the product or its environment that can affect the certified properties a partial or full re-certification will be triggered.

Overall, while software security certification schemes play a vital role in increasing the trustworthiness of software products, the existing schemes still

suffer from a number of weaknesses. High among these weaknesses is the lack of adaptability and centralization.

The fast-paced and dynamic nature of the technological landscape, coupled with the ever-evolving classes of threats, poses a considerable challenge for traditional certification schemes. The lack of a dynamic and fast way to incorporate up-to-date security practices and new attack vectors opens a significant gap in ensuring the effectiveness of the certification process.

Additionally, software security certification can greatly benefit from a decentralized approach. Decentralizing the certification process will allow independent actors with different expertise and background to collaborate and work together in evaluating and certifying software security. Thus, enhancing the robustness and effectiveness of the process.

Chapter 3

Part 2: Evidence-based Certification of Cloud Services

Certification schemes for Cloud extend the traditional ones with the primarily scope of making them applicable in the dynamic distributed environment such as the Cloud following three principal directions: i) flexibility of the certification process (i.e., models for certification life cycle, target and process), ii) adaptability to service evolution and environmental changes (i.e., incremental certification), iii) feasible chain of trust lowering the CA involvement. In the following we fist present relevant certification schemes (Section 3.1), our scenario we used throughout the book as running example (Section 3.2), the certification building blocks (Section 3.3) and then details on how to model a continuous certification process considering a dynamic life cycle and evolving services (Section 3.4). We then present the problem of incremental certification to cope with evolving services (Section 3.5) and a Test-based Certification Framework (Section 3.6). Concluding this chapter we present the certificate chain of trust required to address the need of trustworthiness of the certification process (Section 3.7).

3.1 Relevant Certification Schemes

Actual certification schemes focused on the certification of service/based and cloud/based systems in virtually any domains (e.g., [6, 7, 8, 9]). Two main approaches to evidence collection have been proposed to support existing certification schemes:

1. *test-based evidence collection*, where evidence is collected as the result of testing activities performed by the Certification Authority (CA) on the target of certification [7, 8, 9];

2. *monitor-based evidence collection*, where evidence is collected in the form of metrics retrieved by monitoring service execution [6, 10, 11, 12, 13, 14, 15, 16, 17].

These approaches have been then applied in additional scenarios, including compliance with Service/Level Agreements [11, 18, 15, 14, 19, 20, 21], verification-/as/a/service in grid computing [22], behavioral analysis of Network Virtualization Functions [23], as well as data integrity in cloud/edge datastores [24]. Peculiar schemes also include other means for evidence collection, such as Trusted Platform Modules [25], and formal methods for certification of *functional* properties (i.e., correctness) [22, 26]. Other certification schemes are supporting self/adaptive systems, ensuring the compliance of a system and driving the system adaptations according to properties in the certificates [27, 28, **?**, **?**].

3.2 The Reference Scenario for Certification

In the following we describe a compliance framework we used as reference scenario for the entire book. Our compliance framework is cloud native and capable to execute assessment activities on a target system to evaluate the level of compliance to a specific regulation. It has to be adopted as part of life cycle of the target of the compliance evaluation, and therefore included in its DevOps pipeline. It is made of a composition of services implementing specific functionalities needed for compliance evaluation. Given the sensitivity of the activities carried out and data managed it is mandatory that itself shows some non-functional properties such as confidentiality, integrity and privacy to name but a few.

Figure 3.1 shows the architecture of the assurance framework based on *i)* compute nodes executing evaluation activities on different hooks (*Execution Manager*) and verifying compliance according to collected artefacts (*Compliance Manager*), *ii)* storage nodes storing the compliance controls (*Compliance Control Repository*) and the results of compliance verification (*Artefacts Storage*). The compliance engine implementing this architecture is part of the application life cycle as first class citizen, being able to scale and migrate with the application. It exposes APIs that are called during the different stages of DevOps.

The *Execution Manager* is in charge of the compliance controls. It must be included into the automatic application deployment procedure ensuring that the hooks of the target application are always reachable. This includes replicating and migrating the Execution Manager together with the application, and elastically scaling based on the observed compliance load in terms of number of controls to be simultaneously executed.

The *Compliance Control Repository* includes a set of versioned compliance controls (e.g., scripts) that can be executed by the Execution Manager. Each control is identified by an unique name, a version, and requires a number of parameters for setting up the verification. Its output includes a set of artefacts (e.g., relevant chunks of configuration data) represented in a machine-readable

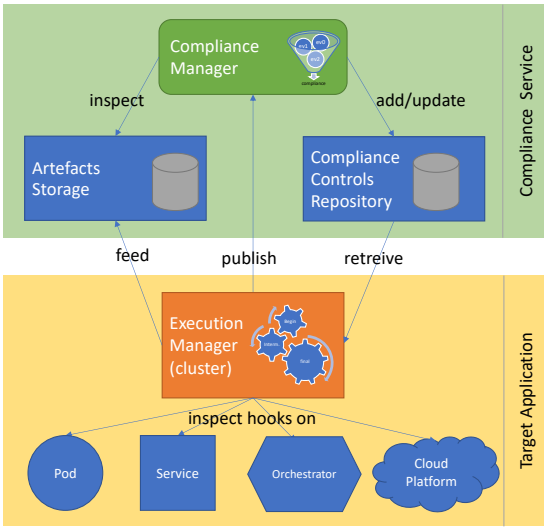

Figure 3.1: The architecture of our reference scenario.

format. For instance, a control can verify whether a banning service connected to an access control system works correctly. To do so, the control can access multiple times, on behalf of a user, a resource without holding the access privileges. If the corresponding user is banned the control returns a positive result, otherwise a negative result.

Hooks define how compliance controls connect to the target of evaluation and the credentials needed to access it. The *Artefacts Storage* collects the results of the compliance control execution. The *Compliance Manager* is responsible to verify compliance by evaluating the collected evidence. It is also responsible to add/update the compliance controls in the Compliance Control Repository. It is connected to the Execution Manager for run-time reconfiguration.

This framework, presented in detail in [29], permits also to keep controls dependencies confined and preserve isolation.

3.3 Certification Building Blocks

In the following we describe the building blocks of a certification process and more specifically the notion of i) non-functional properties as the scope of the certification assessment, ii) non-functional mechanisms as the software components supporting a given non-functional property, iii) Target of Certification as the perimeter of the certification in terms of the set of mechanisms to be verified and iv) evidence as the concrete artefacts of a certification process.

3.3.1 Non-Functional Properties

Assessing if a given non-functional property holds for a given system is the final scope of every certification scheme. The notion of non-Functional property is quite well known in the research community [30, 31, 32, 33, 34] and largely adopted for building domain-specific vocabularies for regulations (e.g., [**?**]), standards (e.g., [**?**]) and cloud security specifications (e.g., [**?**]). In the following we provide an informal definition of generic non-functional properties.

- *Confidentiality* is the capability of limiting information access and disclosure to authorized clients only.
- *Integrity* is the capability of preserving structure and content of information resources.
- *Authenticity* is the capability of ensuring that the clients or objects are genuine.
- *Non-repudiation* expresses the capability of preventing denial of having sourced an information resource.
- *Robustness* is the ability of a computer system to cope with errors during execution or the ability of an algorithm to continue to operate despite abnormalities in input, calculations, and the like.
- *Availability* is the capability of guaranteeing continuous access to data and resources by authorized clients.
- *Performance* is the capability of guaranteeing specific performance like response time or data ingestion latency.

Since these properties represent generic security requirements and provide no information on how to achieve them we deemed them as "abstract" non-functional properties. In order to specify an abstract non-functional property into what we called a "concrete one", it can be complemented by attributes as follows.

Definition 3.3.1 (Non-functional Property) *A non-functional property p can be defined as a pair $(\hat{p}, A_p)$, where $\hat{p}$ is an abstract property and A_p is a set of class attributes specifying it.*

Attributes can include information on the non-functional mechanisms guaranteeing $\hat{p}$ (e.g., access control), a level modeling property strength (e.g., low, medium and high similarly to the CVSS score), and contextual attributes (e.g., *uptime = 98%* for property availability).

Non-functional properties can be organized into an hierarchy $\mathcal{H}_P=(\mathcal{P},\preceq_P)$ [35], where $\mathcal{P}$ is the set of properties and $\preceq_P$ is a partial order relationship over $\mathcal{P}$. Given two properties $p_i,p_j\in\mathcal{P}$, p_i is weaker than p_j (denoted $p_i\preceq_P p_j$) if $p_i.\hat{p}=p_j.\hat{p}$, and $\forall a_k\in A_p$, either the value $v_i(a_k)$ of attribute a_k is not specified or $v_i(a_k)\preceq_{a_k} v_j(a_k)$. We note that total order relations $\preceq_{a_k}$ between contextual attributes $a_k\in A_p$ can be defined by expert users.

3.3.2 Non-Functional Mechanisms

A non-functional mechanism θ is software component that supports a given (set of) non-functional property (e.g., encryption mechanism supporting integrity). It can be formalized as a pair $(\hat{\theta},A_m)$, where $\hat{\theta}$ is a mechanism type (e.g., digital signature), and A_m is a set of attributes specifying configurations affecting the mechanism behavior (e.g., *standard* = *ECDSA*). Non-functional mechanisms can be organized in terms of their type $\hat{\theta}$ into an hierarchies $\mathcal{H}_{M_i}$. This hierarchy $\mathcal{H}_{M_i}$ is defined as a pair $(\mathcal{M}_i, \preceq_{M_i})$, where $\mathcal{M}_i$ is the set of mechanisms of the same type, and $\preceq_{M_i}$ is a partial order relationship over $\mathcal{M}_i$ normally defined by domain experts.

For instance given two mechanisms θ_j,θ_k, θ_j is weaker than θ_k (denoted $\theta_j \preceq_{M_i} \theta_k$) if $\theta_j.\hat{\theta}=\theta_k.\hat{\theta}$, and $\forall a_t \in A_m$, either the value $v_j(a_t)$ of attribute a_t is not specified or $v_j(a_t) \preceq_{a_t} v_k(a_t)$ where the total order relations $\preceq_{a_t}$ between attributes $a_t \in A_m$ are normally defined by expert users. For the sake of completeness, a special hierarchy $\mathcal{H}_{M_f}=(\mathcal{M}_f,=)$ where $\mathcal{M}_f=\{(Functional,\{\})\}$ is also defined in order to model the functional mechanisms of a given service. Both functional and non functional mechanisms can be logically seen as part of a common hierarchy $\mathcal{H}_M$ of mechanisms $\mathcal{M}$ with a common ancestor denoted as *any*.

Each mechanism is also annotated with a set of events ($\{events\}$) or configurations impacting its execution.

3.3.3 Target of Certification

The *Target of Certification* (*ToC*), defines the mechanisms that are part of the certification perimeter for which a given (set-of) non-functional properties applies. We note that in many situations, to support a given non-functional property one or more cooperating mechanisms in the *ToC*are needed.

ToC can be formally defined as (Θ,b), where $\Theta=\{\theta_i\}$ is a set of mechanisms $\theta_i \in \mathcal{M}$ and b specifies the provisioning layer for the system to be certified (i.e., service, platform, infrastructure) .

For instance, let us consider a *ToC* with a binding defined at service layer (i.e., b=<*service*>) to be certified for security property p = (*Confidentiality*, {`ctx`=*in-transit*/*at-rest*}). The *ToC* includes two mechanisms θ_1 and θ_2 supporting p. More specifically mechanism θ_1=(*encryption*,{`algo`=*XML-encryption*, `protocol`=*WS-Security*, `level`=*message-in-transit*}) refers to a mechanism implementing an encrypted communication channel, mechanism θ_2 = (*encryption*, {`system`=*encrypted DBMS*}) identifies a mechanism implementing an encrypted DBMS.

3.3.4 Evidence

Evidence *ev* is the artefacts collected by the certification process in order to verify is a given non-functional property is supported or not. In order to collect evidence each mechanisms θ in the *ToC*can be further modelled in terms

of its behaviour. Solutions based on Symbolic Transition Systems (STS) [35] have been proposed in literature aimed at providing an accurate model to the accredited labs for the generation and execution of evidence collection activities. Evidence supporting the Certification process can be of different types.

Schemes using monitoring evidences like the ones used in the FP7 CUMULUS project [36] are based on i) rules expressing conditions that must be satisfied during the monitoring of a given ToC, ii) monitoring assumptions specify how to record and update variables indicating the state of ToCduring monitoring. They normally use languages based on Event Calculus [**?**] for expressing monitoring conditions (e.g., $EC\text{-}Assertion+$). This approach is used in the EVEREST monitoring system [**?**]. Evidence in these schemes are the monitoring events and the period of evaluation have to be indicated. To be verified such evidences have to specify the minimum period of monitoring ToC, the minimum number of monitoring events and their representativeness.

Schemes based in Trusted Computing initially proposed in the FP7 CUMULUS project requires that the ToC resides on an hardware having TPM modules [**?**] installed. These schemes are very trustworthy given the use of TPM but can support just few properties like software integrity and in most of the cases just at the lowest layers of the Cloud Stack. Given the nature of TC, the trust chain starts with the trust on the TPM and physical platform hosting the TPM chip. It is capable to measure the integrity of the firmware and software in the upper stack reaching the applications layer but in most of the cases requires the support for virtualized TPMs [**?**]. The assessment scheme of TC-based certification is based on *remote attestation* of ToC integrity which is a functionality provided by trusted computing platforms. TC solutions are also used in the context of certification of a way to verify integrity of the evidence collected using other schemes like the testing or monitoring ones.

In COMMON CRITERIA as well as in FP7 ASSERT4SOA project [37], in addition to test-based evidence also formal proof evidence are considered. In the case of formal proofs, the evidences are the results of the execution of the proves on the target system [38]. As in COMMON CRITERIA such evidence are the mostly trustworthy ones and can be applied just in some specific scenarios and for some properties, and having the full access to the system source code in order to build the formal model.

Being the mostly largely used, in the following we concentrated on Test-based evidence only.

Test-based Evidence

Most of the testing-related literature has considered the problem of testing to assess the correct functioning of a service and to automatically generate test cases used in the verification process [39].

We can consider three main *categories* of tests, each of them further specified in terms of *test types* that specify the test cases needed to support security properties on a given service have been generated. More specifically:

- Functionality category: including functional tests based on valid input. It distinguishes between three test types: i) *input partitioning* where testing activities are based on different input partitioning strategies(i.e., *Random Input*, *Boundary Value*, *Equivalence Partitioning*, and *Fuzzy/Mutation*); ii) *model control flow* where testing activities are based on path analysis performed on the service model; iii) *code walkthrough* where testing activities are based on manual analysis of the source code by experts.

- Robustness category: including tests based on invalid and malformed input. It also includes stress and load tests.

- Penetration category: including tests based on well-known security attacks. It considers test types in the form of attacker capabilities (e.g., add, modify, copy)

We note that functionality and robustness test categories are needed in order to check functional correctness and robustness of a given mechanism θ. Test categories and types, according to [**?**] can be organized into hierarchies meaning that, assuming the same certified property and test category, a certificate with a given test type can be "stronger" than a certificate with another test type.

Test based evidence *ev* can then be formally defined as follows.

Definition 3.3.2 (Test-based Evidence) *Let cat(.) be a test category, type(.) a test type, ta(.) a set of test attributes, tc(.) a set of test cases, and tr(.) the results of test case execution. An evidence, denoted ev_i, is a set of 5-tuple $\langle cat(ev_i)$, $type(ev_i)$, $ta(ev_i)$, $tc(ev_i)$, $tr(ev_i)\rangle$.*

A test case *tc*(.) can be further specified in terms of preconditions (*Pr*), inputs (*I*), expected outputs (*EO*), and postconditions (*Po*). It is normally designed for a particular objective, such as to exercise a particular execution path or to verify compliance with a specific requirement. A test case *tc*(.) can be formally defined as follows.

Definition 3.3.3 (Test case) *A test case tc(.) is a 5-tuple $\langle \theta$,* Pr,HC,{(I_1, EO_1), ...,(I_n,EO_n)},Po$\rangle$, *where θ is the mechanism under evaluation,* Pr *is a set of pre-conditions,* HC *a set of hidden communications,* {(I_1,EO_1), ...,(I_n,EO_n)} *a set of pairs (input,expected output), and* Po *a set of post-conditions.*

Test cases can be specifies for the following purposes:

1. *Operation-oriented test cases* (*OO*). Focused on testing each single service operation. Each test case *tc*(.) includes a pair (*I*,*EO*), with *I* the input and *EO* the expected output related to a specific service operation.

2. *Workflow-oriented test cases* (*WO*). Focused on testing multiple operations in a workflow fashion. Each test case *tc*(.) defines a specific path in the workflow to be tested and is composed of a set of pairs (I_{o_j},EO_{o_j}), with o_j the *j*-th operation in the workflow. We note that, when the output of a given operation is not relevant to the testing activity, EO_{o_j}=*null*.

3. *Implementation-oriented test cases* (IO). Focused on testing internal states of a given operation endpoint. It requires full access to the source code or a detailed model of it. Form a praxctical point of view they are similar to the workflow-oriented test cases, each test case is composed of a set of pairs (I_{o_j},EO_{o_j}). In this case, however, the service implementation is also considered to generate the test cases and to test specific flows to covel internal implementation-specific branches.

3.4 Certification Modelling

Certification schemes for cloud services relies on three modelling layers:

- **Mechanism Model:** it describes how to model the mechanisms θ in ToC including the visibility level from white to black box. It is fundamental to drive evidence collection process. In general the more the model is accurate the better the evidence collection process can be instrumented. Model quality metrics can be defined with the scope of understand the level of details available to carry our certification.

- **Certification life cycle Model:** It describes how to model the evolution of certificate across time, from its issuing to possible expiration or revocation. The CA is traditionally responsible to take decisions on the certificate life cycle like certification issuing, suspension, revocation, or expiration for instance, as a reaction to new threats, regulations or audit activities.

- **Certification Process Model:** It describes how to model the entire certification process including all the building blocks in Section 3.3. The certification process is responsible for all evaluation activities aimed to produce a certificate for the ToC (*issuing phase*), as well as to continuously verify the validity of the certificate against context changes to reduce unnecessary certificate revocation and re-certification [**?**] (*post-issuing phase*). It specifies how the evidence are collection and evaluated. Evidence quality metrics can be specified in order to evaluate the strength and the trustworthiness on the certificate.

We note that certification modelling depends on the evidence type, in the following we concentrated on modelling suitable for testing evidence. Software mechanisms modelling is largely covered in literature in the contest of normal software testing and evaluation practices []. In the following we concentrated on the last two models starting from the Certification life cycle model.

3.4.1 Certificate Life Cycle Model

Cloud-related certificate must evolves in time following the evolution of the target service, threats and regulations. This evolution is much more dynamic than for traditional software and thus have to be automatized as much as possible.

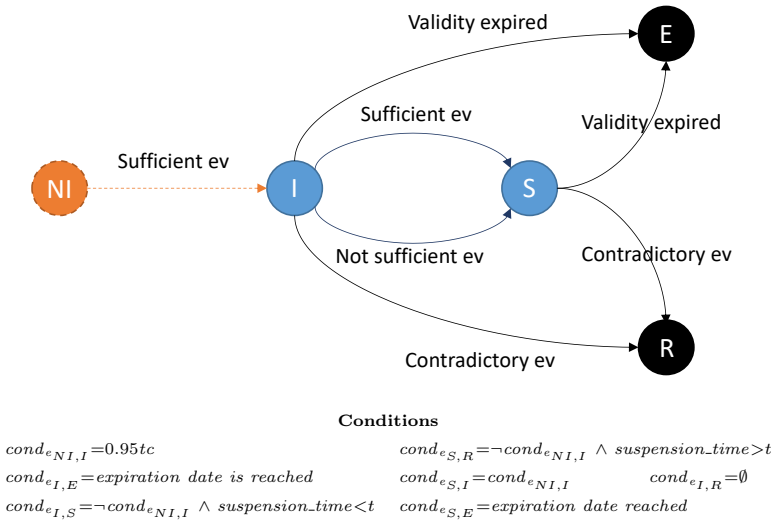

Figure 3.2: Life cycle with states Not Issued (NI), Issued (I), Suspended (S), Expired (E), Revoked (R) and examples of conditions on transitions.

For this reason a certificate life cycle l is defined having the scope of modelling the certificate evolution from its issuing to possible expiration or revocation. It is handled i) offline by the *CA* in a static and asynchronous way as a reaction to new threats and regulation changes and ii) online and run-time on the basis of evolving evidence by the accredited lab since the static intervention of a *CA* is not always feasible.

Definition 3.4.1 (life cycle) *The life cycle l is modeled as a deterministic finite state automaton $G^l(V^l,E^l)$ where each vertex $v \in V^l$ represents the certificate state (e.g., not-issued, issued, suspended, revoked) using a label $label(v)$ and each edge $e=(v_i,v_j) \in E^l$ represents the transition between two states using a labeled representing the condition $cond_e$ over certificate evidence that regulates the transition.*

Figure 5.2 shows an example of the life cycle automaton with transition conditions. For instance, edge $e_{NI,I}$ is labeled with a condition $cond_{e_{NI,I}}$ stating that at least 95% of the test cases $tc(.)$ need to be successfully executed (i.e., $cond_{e_{NI,I}}$=0.95tc) for the certificate to move from state NI to state I.

3.4.2 Certification Process Model

In the following we first present our conceptual framework supporting the certification process and then the model of our certification process incarnating our conceptual framework.

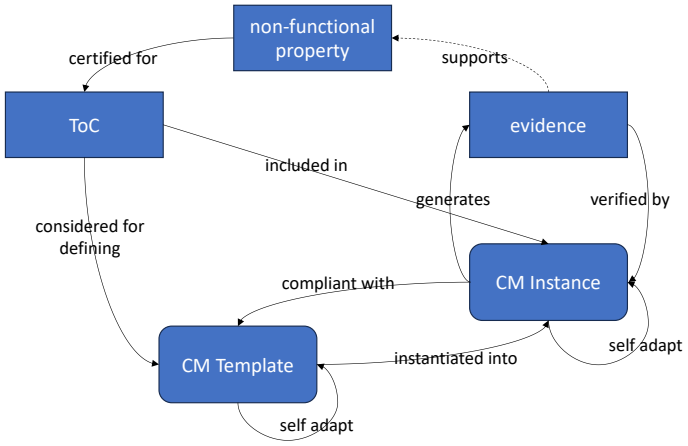

Figure 3.3: The conceptual framework.

Conceptual framework

To model certification process three aspects need to be considered, i) the abstract declarative model defined by the CA describing the property p to be certified, the target of the certification activities ToC and evidence to be collected, and the evidence collection activities to be executed (Certification Model (CM) Template), ii) the procedural model of the concrete activities to be carried our by the accredited lab on the given implementation of the target (Certification Model (CM) Instance), iii) the model of the certificate to be released. Figure 3.3 shows the conceptual framework at the basis of the certification process.

The aim of a certification process is to prove a property p for a given ToC. It is driven by a CM Instance. CM Instance is generated as a concrete incarnation of a given CM Template and verified against it to ensure its validity. The evidence is continuously collected, according to the CM Instance collection procedure. In case the evidence is not sufficient to prove a given non-functional property (dashed arrow in Figure 3.3) the corresponding certificate cannot be awarded.

The CM Template $\mathcal{T}$ is defined by the CA and drives the entire certification process. It represents the cornerstone of the certification chain of trust which is grounded on the correctness of the certification process itself as it is designed and signed by the CA (see Section 3.7). It is a declarative high-level representation of the process including requirements specification, configurations, and activities for the certification of a property for a given (class of) ToC.

Definition 3.4.2 (CM Template $\mathcal{T}$) *A CM Template $\mathcal{T}$ is defined as 5-tuple of the form $\langle p, ToC, m, ev, l \rangle$ and signed by the CA. The requirements on the ToC are expressed in the CM Template $\mathcal{T}$ in terms of the mechanisms to be used to support a given property p. The activities for the certification are expressed*

using a model m describing the evidence collection, and the evidence ev to be collected and the certificate life cycle l according to them.

More specifically the evidence collection model m describes the set of execution flows $\Phi(m)$ targeting the mechanism in Θ, which must be evaluated to certify p. The flows are structured to cover the different relevant purposes of testing (i.e., operation OO, workflows WO and Implementation IO, see Section 3.3.4) Evidence collection flows $\Phi(m)$ are associated with evidence ev to be produced. The evidence ev in $\mathcal{T}$ is expressed in an abstract format indicating for instance the inputs I_n and expected outputs EO_n in the form of partitions of the corresponding input and output domains (see Definition 3.3.3). The life cycle l is also part of the CM Template. It is defined by the CA to describe the certificate evolution across time depending on the outcomes of the evidence collection and other external events like expiration and revocations. We note that at CM Template level, the mechanisms $\theta \in ToC$ can be defined at very abstract level just including the type only (i.e., $(\hat{\theta},\emptyset)$) or at a more concrete level including specific attributes (i.e., $(\hat{\theta},A_m)$).

Example 3.4.1 ($\mathcal{T}$) *Let us consider the Artefacts Storage service of our scenario in Section 3.2 an the property* Confidentiality *for which it has to be certified. A CM Template $\mathcal{T}$ includes the+ security property p = (*Confidentiality*,* {`ctx` = in-transit/at-rest}*) and target of certification ToC= (*{θ_1, θ_2}, <service>*),* *where θ_1 =(*encryption, {`level` = internal-communications}*) is an encryption mechanism securing internal service communications, θ_2 =(*encryption, {`level` = data-at-rest}*) is an encryption mechanism securing stored data. Evidence collection model m in $\mathcal{T}$ specifies the flows of execution, by combining mechanisms in ToC. Evidence ev in $\mathcal{T}$ is expressed in terms of generic test category and types and using partitions for input and output instead of real data. The life cycle l_is fully defined by the CA in terms of states and transition conditions specified in terms of evidence aggregations.*

CM Instance $\mathcal{I}$ is the concrete implementation of a given CM Template $\mathcal{T}$. It includes details on configurations and activities to be executed on the ToC with the final scope of collecting evidence. CM Instance $\mathcal{I}$ is an join effort between the accredited lab and the service provider contributing to the establishment of a full chain of trust (see Section 3.7). Before executing the CM Instance $\mathcal{I}$, the accredited lab verifies *i)* the signature of the $\mathcal{T}$from which it is generated or to which is claiming conformance, *ii)* the correctness of the instantiation procedure and *iii)* The correct representation of the ToC (see Section **??**).

It is defined as follows.

Definition 3.4.3 (CM Instance $\mathcal{I}$) *A CM Instance $\mathcal{I}$ is a 5-tuple of the form $\langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l} \rangle$ it represents the implementation of a given CM Template $\mathcal{T}$ which is signed by a CA. It contains i) $\overline{ToC}$ as the concrete incarnation of a given ToC, where all the configurations for the real implemented mechanisms execution are specified, ii) the evidence collection process $\overline{m}$ that can be executed thanks to the*

*configuration in $\overline{ToC}$ iii) thee resulted evidence $\overline{ev}$, iv) the certificate life cycle $\overline{l}$ evaluating the evidence in relation with the certificate status (see Section **??**).*

We note that the concrete evidence $\overline{ev} \in \mathcal{I}$is defined in terms of category and types and in terms of real input and expected output in conformity with the partitions expressed in $ev \in \mathcal{T}$. The evidence $\overline{ev}$ also refers to the real mechanisms $\theta \in \overline{ToC}$ so that, executing the concrete collection model $\overline{m}$, they can be re-evaluated if needed (e.g., in case of re-certification).

Example 3.4.2 ($\mathcal{I}$) *Let us consider CM Template $\mathcal{T}$ in Example 3.4.1. A CM Instance $\mathcal{I}$ for $\mathcal{T}$ can shows the following elements. Security property $\overline{p}$=(*Confidentiality,{`ctx` = in-transit/at-rest}*). Target of certification $\overline{ToC}$= ($\{\theta_1, \theta_2\}$,* <service>*), where θ_1 =(*encryption,{`algo` = ssh, `protocol` =TLS1.3, `level` = message-in-transit}*) implements a secure channel mechanism based on TLS1.3, θ_2 = (*encryption, {`level`=at rest, `algo` =encrypted DBMS}*) implements an encrypted DBMS for artefacts storing. The remaining components should be such that:* i) $\overline{m}$ *is consistent with m in $\mathcal{T}$,* ii) $\overline{ev}$ *reflects the description in the ev of $\mathcal{T}$,* iii) $\overline{l}$=l.

The final outcome of a certification process is a certificate. It plays a fundamental role for improving the trustworthiness of cloud services/systems adoption and it is an enabler for advanced processes such as certification-aware service selection/discovery and composition, to name but a few. A certificate $\mathcal{C}$ is a 4-tuple $\langle \mathcal{I}, \mathcal{T}, ws, \overline{ev}_r \rangle$, where $\mathcal{I}$ is a CM Instance, $\mathcal{T}$ the Template used to instantiate the $\mathcal{I}$, *ws* service endpoint where certificate is attached to, and $\overline{ev}_r$ the results of the execution of test cases in $\overline{ev} \in \mathcal{I}$.

We note that according to Figure 3.3, both the CM Template and the CM Instance are capable to adapts themselves to changes supporting incremental certification (see Section 3.5).

The Process

Figure 3.4 shows our certification process, which is based on the conceptual framework in Figure 3.3. It is concretely implemented composed of five main steps.

1. *CM Template Definition*: the CA designs a CM Template $\mathcal{T}$ detailing the methodology for certifying p on a class of *ToC*. We note that this definition might be carried out well before the execution of the other steps of the certification process. It ease the involvement of the CA in dynamic cloud certification since its intervention it is decoupled from the process execution.

2. *Service Implementation*: the service provider implement the service *ws* to be certified eventually considering the CM Template as a guide for a certification-oriented design.

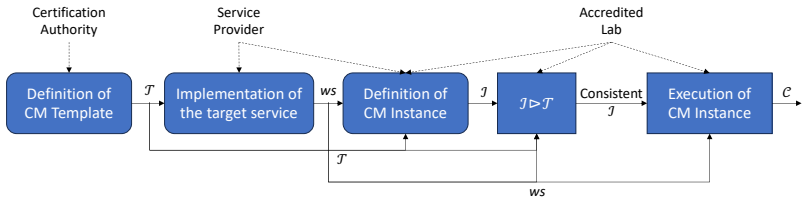

Figure 3.4: Certification process: execution steps. Rounded boxes represent steps requiring tools or guidelines for service providers, accredited labs, or CA. Squared boxes represent concrete executable tools that have to be implemented.

3. *CM Instance Definition*: the accredited lab, with the help of the service providers, produces CM Instance $\mathcal{I}$ for the implemented service using the CM Template as a guide to be compliant with. We note that CM Instance can also be defined independently, but have to show conformance to a specific CM Template.

4. Consistency check ($\mathcal{I} \triangleright \mathcal{T}$): the accredited lab verifies the consistency between $\mathcal{I}$ and $\mathcal{T}$. This is crucial to build the certification chain of trust for the cloud. It verifies whether a CM Instance $\mathcal{I}$ is a correct and valid instantiation of a CM Template $\mathcal{T}$. This step is mandatory both when $\mathcal{I}$ is generated independently by service provider owning the ToC, or if it is generated with the support of the accredited labs.

5. *CM Instance Execution*: the accredited lab execute all certification activities aimed to first certificate issuing (*issuing phase*) and then certificate adaptation (*post-issuing phase*).

We note that the proposed certification process improve trustworthiness via delegation of trust to the accredited lab and supporting consistency check function $\triangleright$ verifying the compliance between what was signed by the CA and executed by the accredited lab. It also provide a dynamic and run-time certificate life cycle management suitable for the cloud.

3.5 Incremental Certification

The first certificate issuing is normally executed in a laboratory environment under the supervision of the CA. In some specific situations it can be executed also in operation environment by the delegated accredited lab (e.g., in case the property depends on some operation systems that cannot be replicated in laboratory). Incremental certification operate after the certificate issuing focusing on updating certificate, re-executing part of the certification process if needed. It is clear that incrementalty is a crucial property for a certification scheme for the cloud. More specifically incremental certification aimed at maximizing

certificate validity, while minimizing the risk of unnecessary certificate revocation, reducing as much as possible the amount of re-certification activities. A certificate revocation in fact requires re-certification from scratch, which introduces high cost and time overheads invalidating the benefit introduced by cloud certification schemes.

3.5.1 Certificate Adaptation

To support incremental certification, after the certificate issuing both the CM Instance and CM Template can adapt themselves to different types of changes effecting cloud environment.

We consider two adaptation scenarios: *i) CM Instance adaptation* in case of service, platform, or infrastructure changes, or any change in the configurations at layers specified in the *ToC* for instance to cope with elastic scaling or migration; *ii) CM Template adaptation* in case of new regulations or requirements to assess the validity of a given property (e.g., a new issues affecting a mechanism in the *ToC* or a new threats). We note that any change in CM Template also triggers an adaptation process on relative CM Instance. Certificate adaptation provides the ability to re-execute (part of) the process in Figure 3.4, following changes in the CM Template, the CM Instance, and the service implementation.

CM Instance Adaptation

Let us consider an adapted CM Instance $\mathcal{I}'= \langle \overline{p}', \overline{ToC}', \overline{m}', \overline{ev}', \overline{l}' \rangle$ of $\mathcal{I}= \langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l} \rangle$. We consider three possible reaction to changes at $\mathcal{I}$ level:

Partial re-evaluation. It applies when the modified $\mathcal{I}'$is still consistent with the original $\mathcal{T}$. It requires the execution of a partial evidence collection process based on the differences between $\mathcal{I}$ and $\mathcal{I}'$. If possible, it leads to certificate renewal according to the following scenarios.

- *Cloud event.* The occurrence of an event affects a mechanism $\theta_i \in \overline{ToC}'$ of $\mathcal{I}$. Test cases involving θ_i are executed again.
- *Additional tests.* Additional test cases added in $\mathcal{I}'$, due to a change in $\overline{ToC}'$, $\overline{m}'$, or $\overline{ev}'$, and executed.
- *New mechanism.* A mechanism replacement between a new mechanism $\theta_j \in \overline{ToC}'$ of $\mathcal{I}'$ and an existing mechanism $\theta_i \in \overline{ToC}$ of $\mathcal{I}$ having the same type $\theta_j.\hat{\theta}=\theta_i.\hat{\theta}$. Test cases involving θ_i in $\mathcal{I}$ are executed again according to the new $\overline{ToC}'$ and $\overline{m}'$.

If enough correct evidence is collected the certificate returns to issued state. On contrary, partial re-certification or revocation can be triggered. We note that, since the authority intervention is not needed, *partial re-evaluation* can be executed at run-time following $\mathcal{I}'$.

Partial re-certification. It applies when the modified $\mathcal{I}'$is no more consistent with the original $\mathcal{T}$. It requires the execution of a partial evidence collection process that first of all searches for a new $\mathcal{T}'$ showing consistency with $\mathcal{I}'$ and then follows the differences between $\mathcal{I}$ and $\mathcal{I}'$The focus of partial re-certification, rather than implement a complete re-certification from scratch, is to exercise the flows in $\mathcal{I}'$ that either do not exist or differ from the ones in $\mathcal{I}$. New test cases are then executed to collect the evidence needed to award a certificate for a new property $\overline{p}'$ in $\mathcal{I}'$. If the evidence is both correct according to $\mathcal{T}'$, and sufficient according to $\overline{l}$ the certificate is renewed, otherwise the certificate is revoked. We note that as a lightweight alternative CA can define suitable CM Template $\mathcal{T}'$ for the adapted CM Instance $\mathcal{I}'$, such that a weaker property is still preserved for the service referring to it (i.e., Certificate Downgrade []) We also note that a certificate that was downgraded can be then upgraded in case the partial re-certification succeeded.

Full re-certification. It is applied in case changes to $\mathcal{I}$ cannot be managed according to one of the above approaches.

CM Template Adaptation

CM Template adaptation focuses on partial updates of the certification methodology. It is driven by the CA that releases a refined CM Template $\mathcal{T}'=\langle p',\ ToC',\ m',\ ev',\ l'\rangle$ of a given $\mathcal{T}=\langle p,\ ToC,\ m,\ ev,\ l\rangle$, triggering a CM Instance adaptation for all instances $\mathcal{I}$ referring to $\mathcal{T}$.

CM Template adaptation is a way to support certification-aware resilience to changes. For instance, let' suppose that a new vulnerability for a given $ToC$was identified, which requires a modification of $\mathcal{T}$. Such modification triggers an adaptation process requiring that all certificates referring to affected template become immediately *suspended*. The service provider must then adapt the affected services and corresponding instance $\mathcal{I}$ to keep the certificate.

3.6 Certification Framework

In this section we will describe a certification framework implementing the process in Figure 3.4 suitable for an accredited lab. It provides a set of tools and guidelines for designing CM Templates, generating CM Instances, executing consistency check, and in general the certification management. It is based on two main components: *i) Certification Model Manager* for CM generation, selection, verification, *ii) Evidence Collector Manager* to parse the CM Instance $\mathcal{I}$ and drives the evidence collection process by injecting evidence $\overline{ev}$ (i.e., test cases) in $\overline{ToC}$. The proposed framework is generic and can handle different evidence types using type-specific probes to address the need of carry out the inspection (e.g., executing test cases).

Certification Model Manager is the main interface for the accredited labs to manage the certification process. It includes a repository of CM Templates and Instances and supports the building blocks in Sections 3.3 and 3.2, like

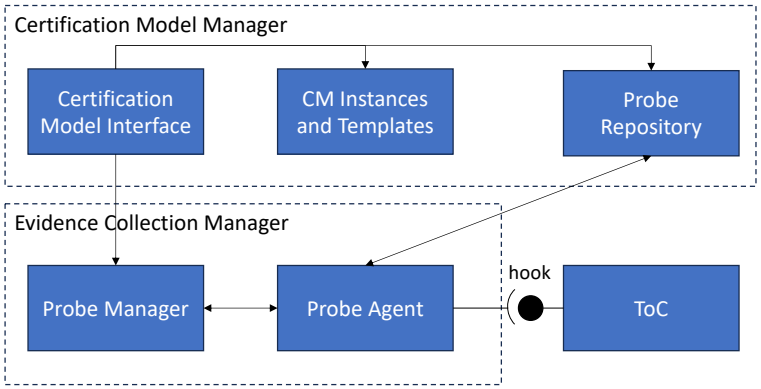

Figure 3.5: Certification framework architecture.

the hierarchies of properties $\mathcal{H}_P$ and mechanisms $\mathcal{H}_M$. It also includes the *Probe Repository*, where all the probes (e.g., test cases) and the corresponding evidence are stored and kept up to date. It offers internal functionalities such as the consistency check function $f^{\triangleright}$ and the certificate adaptation procedures to support the management of the certification process and certificate life cycle (step $\mathcal{I} \triangleright \mathcal{T}$ in Figure 3.4).

Evidence Collector Manager implements the evidence collection architecture driven by CM Instance $\mathcal{I}$ [40] (step *CM Instance Execution* in Figure 3.4). It is composed of a *Probe Manager* (PM), the owner of the collection process, and one or more *Probe Agents* (PAs), responsible for verifying specific flows $\Phi(\overline{m})$ over $\overline{ToC}$. PM *i)* provides interfaces to manage the collection process, *ii)* configures the evidence collection process and *iii)* initializes PAs according to configurations, models, and probes (e.g., test cases) in $\mathcal{I}$, and *iv)* manages the certificate life cycle according to $\overline{l}$. PA uses the *Probes Repository* to retrieve the relevant probe in $\overline{ev}$ and executes them on the ToC, returning the results of their execution to PM.

Figure 3.5 shows the data flow between *Certification Model Manager* and *Evidence Collection Manager*. *Certification Model Manager* is externally accessed through a restful interface, which permits to manage CM Templates and CM Instances, and communicates with TM and TA(s) in the *Evidence Collector Manager*. A certification process starts when the *Certification Model Manager* sends a valid CM Instance $\mathcal{I}$ to TM. Upon receiving $\mathcal{I}$, TM parses it and forwards each of its elements, except property $\overline{p}$, to TA(s). TA(s) retrieves the test cases in $\overline{ev}$ of $\mathcal{I}$ from *Test Suites Repository* and manages the testing activities. The testing activities are executed by one or more probes that access the ToC through a hook. TA(s) sends collected evidence back to TM when available, which aggregates it and eventually triggers a life cycle transition.

3.7 Certification Chain of Trust

The definition of a proper trust model is fundamental for the usability of a cloud certification process in real scenarios. In Section **??** we define the chain of trust for software certification focused on the role of the certification authorities. In the case of cloud certification the certification authority still a central entity but empowered by a delegated accredited lab capable to execute part of the certification process on behalf of the certification authority addressing dynamicity of the cloud and increasing the confidence of final users in the results of the certification process itself.

In the following we present an detailed description of the chain of trust for cloud service certification.

3.7.1 Chain of Trust

In cloud certification, the certification authorities cannot be assumed as a single trusted entity taking responsibility on (i.e., signing) the whole certification process. There is a clear need to define a chain of trust where responsibilities are spread across the entities involved and the entire life cycle of the certification. We therefore envision a chain of trust based on multiple signatures. Similarly to Section **??**, 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 $A_{en,ws}$. The customer c's trust in an assertion $A_{en,ws}$ made by an entity en is denoted $\tau_c(A_{en,ws})$, where τ_c takes discrete values on an ordinal scale (e.g., for a Common Criteria [41] certified product, an assurance level value in 1-7).

Cloud certification chain of trust is based on three different signing moments, one for each of the fundamental certification models (CM Template, CM Instance, Certificate) as follows.

- *CM Template signature*: Being the declarative description of the methodology for the certification of a class of ToC, the CM Template $\mathcal{T}$ is signed by a trusted certification authority CA. $\tau_{AL}(\mathcal{T}_{CA})$ denotes the trust an accredited lab AL has in CM Template $\mathcal{T}$ that builds on the trust AL has on CA and its signature.

- *CM Instance signature*: Being the procedural description of the certification activities to be executed, the CM Instance $\mathcal{I}$ is signed by an accredited lab AL. The AL is responsible for instantiating the CM Template and for this reason it has been delegated by CA. AL receives a signed CM Template $\mathcal{T}$ and instantiates it (e.g., filling in all missing elements possibly with the help of the cloud/service providers) to form a CM Instance. The CM Instance signature builds on $\tau_{AL}(\mathcal{T}_{CA})$ and is at the basis of the trust $\tau_c(A_{AL,ws})$ and $\tau_c(\ E_{AL,ws})$ a client c has in assertions $A_{AL,ws}$ and evidence $E_{AL,ws}$, respectively, provided by AL.

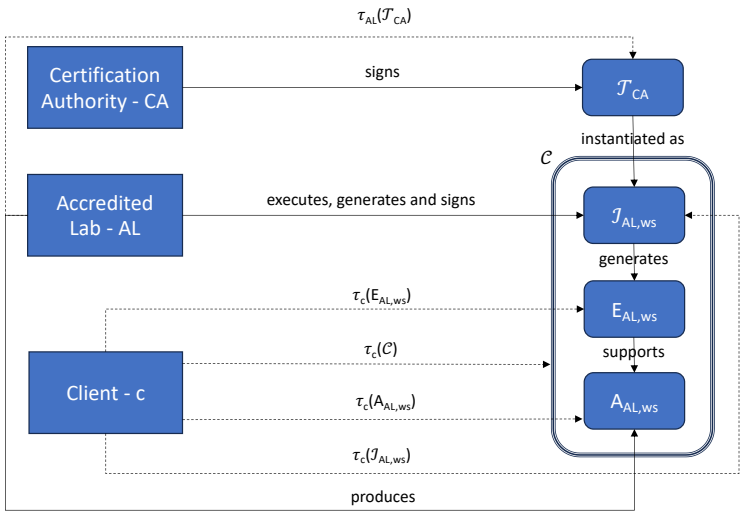

Figure 3.6: Chain of Trust for cloud certification

- *Certificate signature*: Being the final document awarded to the service/system, it signature made by the CA binds assertions $A_{AL,ws}$, evidence $E_{AL,ws}$ and the corresponding CM Instance used to carry out certification activities on the ToC.

Figure 3.6 shows our chain of trust, identifying roles (rectangles), artifacts (rounded rectangles), certification activities (solid arrows), and trust relations (dashed arrows). The chain of trust includes c's trust in *i)* CM Instance $\mathcal{I}_{AL,ws}$, denoted as $\tau_c(\mathcal{I}_{AL,ws})$, used to collect the evidence supporting a set of assertions, *ii)* the evidence generated by AL according to CM Instance $\mathcal{I}$, denoted as $\tau_c(E_{AL,ws})$, *iii)* assertions made by AL on a service, denoted as $\tau_c(A_{AL,ws})$, where $A_{AL,ws}$ is the set of assertions produced by the accredited lab AL on ws, and *iv)* the certificate $\mathcal{C}$ including $A_{AL,ws}$ and $E_{AL,ws}$ and $\mathcal{I}_{AL,ws}$. $\tau_c(\mathcal{C})$ depends on *i)* the reputation of CA signing CM Template $\mathcal{T}$ (i.e, $\tau_{AL}(\mathcal{T}_{CA})$) and the certificate $\mathcal{C}$ itself, *ii)* the reputation of AL and the trust in the methodology used by AL to generate and sign CM Instance $\mathcal{I}_{AL,ws}$ (i.e., $\tau_c(\mathcal{I}_{AL,ws})$), and specify assertions $A_{AL,ws}$(i.e., $T\tau_c(A_{AL,ws})$), and *iii)* the trust in the methodology used by AL to generate evidence $E_{AL,ws}$ (i.e., $\tau_c(E_{AL,ws})$).

3.7.2 Chain of Trust and Life Cycle Management

The chain of trust is meant to support all the phases of the the certificate life cycle in Figure 5.2. Let's start with the issuing phase. It is normally based on static evidence collected and the CM Instance signature follows a two-step process: i) signature of a CM Instance with endpoints that refer to mechanisms deployed in the laboratory, ii) when the certified service ws moved in production

and the certificate $\mathcal{C}_{ws}$ is issued (i.e., transition from NI to I in the life cycle), the CM Instance is substituted with a new one signed by AL with all bindings and endpoints associated with the operation deployment infrastructure.

The chain of trust also supports system evolution and cloud events as discussed in Section 3.5.1. In this scenarios the collected evidence may become insufficient or contradictory, and corresponding certificate invalid, requiring re-certification. The incremental certification process in Section 3.5 focused on renewing a certificate by reusing, as much as possible, the certification evidence available from previous certificates [42], reducing the need of new evidence. The trust in this incremental process is given by the trust $\tau_c(E_{AL,ws})$ the client c has in the evidence E produced by executing CM Instance $\mathcal{I}_{AL,ws}$ and the trust $\tau_c(\mathcal{I}_{AL,ws})$, the client c has in the $\mathcal{I}$. The evolving certificate generated by the incremental process is then handled by the life cycle (e.g., suspension or revocation in the case of insufficient or contradictory evidence). With this chain or trust the involvement of the certification authority is marginal. It needs to sign the adapted certificate if requested by the accredited lab. IT is the accredited lab delegated by the certification authority, which is much more involved in the process. The CA trusts and verifies lab activities by means of digital signature verification. In the case the $\mathcal{I}$ is no more usable for service certification or compliance with the original CM Template is not guaranteed, re-certification from scratch is triggered re-involving the CA.

Chapter 4

Part 3: Certification of Modern Composed Systems

Modern systems are increasingly based on composed nano-services deployed in complex dynamic environment such as the ones based on Edge to Cloud Continuum. In such extremely dynamic and complex scenario it is fundamental to i) increase the quality of the certificate considering multiple factors affecting the final non-functional property to be certified (Section 4.1), ii) handle certification of composition of micro/nano services (Section 4.2), iii) consider peculiarities of the deployment environment while releasing certificates (Section 4.3), and iv) include the development process as a specific target of the certification (Section 4.4). We note that, although among the multiple factors that can be considered in a modern certification process, deployment and development can be partially addressed. However such factors can just partially satisfy the need of having independent and ad hoc certification solutions specialized in deployment environments and development processes.

4.1 Multi-Factor Certification

The unbeatable advantages brought by traditional certification schemes conflict with a strong assumption that limit their quality. Traditional certification assumes certificates awarded to services just according to their software artifacts only, limiting the variety of evidence that can be collected and thus reducing the quality of the certificates.

The certification of modern systems should consider more factors impacting the certificate quality (e.g, information about the programming language used and the type of the development process adopted).

This idea was initially considered by some certification schemes focused on hybrid evidence collection models using combination of testing, monitoring, and formal proof [36]. These schemes although considering evidence of different types improving the variety, were not capable to address different factors being

Table 4.1: Five different implementations s_1–s_5 of the *Artefacts Storage* service of our scenario in Section 3.2. Attributes of a given property "reliability" is specified according to three factors (i.e., traditional software artefacts, the development process and the verification process).

Service	Software Artifacts			Development Process		Verification process	
	Repl.	**Zones**	**HA Prot.**	Prog. Lang.	Dev. Proc.	Trust. Contr.	When
s_1	**3**	**3**	**Managed**	Object-Oriented	Waterfall	No	After
s_2	**3**	**2**	**No**	Object-Oriented	Waterfall	No	During
s_3	**1**	**1**	**Custom**	Imperative	Prototype	Yes	After
s_4	**3**	**3**	**Managed**	Functional and OO	DevSecOps	Yes	During
s_5	**2**	**3**	**Managed**	Imperative	DevOps	Yes	After

concentrated on software artefacts factor only. Modern certification schemes extends the evaluation process of the given *ToC* beyond software artifacts by integrating relevant aspects influencing the certification. For instance peculiarities of the development process has a substantial impact on the resulting software and, in turn, on the non/functional properties it holds [43, 44].

4.1.1 Certification Model

In multi-factor certification schemes, the CM Template ($\mathcal{T}^{fac}$) considers different aspects (factors) influencing the evaluation of the service behavior independently. For example the work in [45] considers three factors:

- Software Artefacts: describing the software artifacts of the target (i.e., traditional certification based on software evidence);
- Development process: describing the development process used to implement the target, such as the programming language used, and the type (e.g., Waterfall, DevOps)
- Verification process: describing additional details of verification process used for the certification such as the fact that it is continuous, based on trusted entities or self generated.

The the multi-factor CM Template $\mathcal{T}^{fac}$ is defined as follows.

Definition 4.1.1 (multi-factor CM Template $\mathcal{T}^{fac}$) *The multi-factor CM Template $\mathcal{T}^{fac}$ is 6-tuple of the form $\langle p, ToC, m, ev, l, \mathcal{F}\rangle$ similarly to the traditional CM Template but extended to cope with the different factors as follows: i) the attribute $A_p \in p$ reorganized in subset of attributes specifying each factor, ii) the ToC reorganized in subset of mechanisms for each specific factor, iii) the evidence collection model m is extended to contain flows $\Phi(m)$ targeting the factor-specific mechanisms in ToC, iv) the evidence ev reflects the changes in m including evidences of different types (e.g., part of configuration files), v) a life cycle l expressing generic transitions among certificate states, vi) a function $\mathcal{F}$ to define how to combine evidence of the different factors (e.g., conjunction, disjunction).*

Table 4.1 shows the multi-factor CM Template $\mathcal{T}^{fac}$ (i.e., just the property p for simplicity) of five different implementations s_1–s_5 of the *Artefacts Storage* service of our scenario in Section 3.2 specifying three factors: software artefacts, the development process and the verification process.

Considering factor software artefacts only both service s_4 and s_1 seems equivalent, being capable to operate with three replicas spread in three zones, with a HA protocol of managed type. Comparing with the others they seem to show the better reliability property. According to traditional certification approaches that does not consider additional factors, they are both awarded with a very similar certificate, where they may differ just in terms of the quality of the collected evidences.

However, considering also the additional factors, it is clear that s_4 is largely better than s_1. According to Table 4.1 s_4 was developed with a probably much robust programming language (*Prog. Lang.* = *Functional and OO*) with a much more advanced development process (*Dev. Proc.* = *DevSecOps*) and verified using a continuous process (*when.* = *During*). We note that, although CM Templates in Table 4.1 already suggest a partial ordering among the CM Templates, the concrete implementations of such Template could refine such ordering specifying better the attributes.

As the traditional certification the multi-factor CM Template $\mathcal{T}^{fac}$ has to be instantiated into a multi-factor CM Instance $\mathcal{I}^{fac}$ in order to be executed.

Definition 4.1.2 (multi-factor CM Instance $\mathcal{I}^{fac}$) *The multi-factor CM Instance $\mathcal{I}^{fac}$ is a 6-tuple of the form $\langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l}, \overline{\mathcal{F}} \rangle$ representing a concrete implementation of a given CM Template $\mathcal{T}^{fac}$. More specifically i) $\overline{p}$ shows concrete attributes to be certified, ii) the $\overline{ToC}$ refers to the real mechanisms for every factor, iii) the evidence collection model $\overline{m}$ refers to factor-specific probes to be executed on the ToC in order to collect, iv) the concrete evidence $\overline{ev}$, v) a concrete factor aggregation function $\overline{\mathcal{F}}$ expressing which evidence of which factor are aggregated and how and iv) the life cycle $\overline{l}$ express state transitions in terms of concrete conditions that are computed based on $\overline{\mathcal{F}}(\overline{ev})$.*

We note that while it can be complex, a simple yet effective evaluation function $\overline{\mathcal{F}}$ can be based on conjunction of all the single-factor evaluations carried out independently [45]. We also note that the concrete property $\overline{p}$ can be fully specified also in the CM Template $\mathcal{T}^{fac}$or further refined in the CM Instance $\mathcal{I}^{fac}$.

Example 4.1.1 (Concrete multi-factor property $\overline{p}$) *Let us consider the Artefacts Storage service s_4 in Table 4.1 where the property p in the multi-factor CM Template is expressed as $p = (\hat{p_{rel}}, (A_p^{art}, A_p^{dev}, A_p^{eval}))$. We note that the attributes $(A_p^{art}, A_p^{dev}, A_p^{eval}$ are subject to total order relations $\preceq_{a_k}$ between contextual attributes $a_k \in A_p^j$ of a given factor j. This total ordering can be further refined in the corresponding concrete property $\overline{p} \in \mathcal{I}^{fac}$. For instance an instantiation of the attribute Prog. Lang. (pl) of factor development process (A_p^{dev}) is subject to a total order where [Rust $\preceq_{a_{pl}}$ Java $\preceq_{a_{pl}}$ Python].*

We note that each factor-specific evaluation in $\overline{m} \in \mathcal{I}^{fac}$ depends on the factor peculiarities as well as on the target mechanisms. For instance, evaluation of software artefacts are carried our as in the traditional evaluation process. Evaluation of the development process is carried out inspecting metadata description attached to the service to be certified, where the attributes of the development process are detailed (e.g., programming language used, the type of process). At this level such evaluations can be expressed as reference to a specific probe executing the evaluation.

Example 4.1.2 (Concrete target $\overline{ToC}$ and evidence collection model $\overline{m}$) *Let us consider the CM Template of Artefacts Storage service s_4 in Table 4.1. The target $\overline{ToC}$ can be defined as = { θ_{art}, θ_{dev}, θ_{eval} }, where for instance $\theta_{art}.A_m$ =* {Replica Manager = *Kubernetes*}*, $\theta_{dev}.A_m$ =* {Pipeline = *File Content,* Source Code = *Rust,* Code Review Document = *File Content*}*. The $\overline{ToC}$ expresses that s_4 is deployed in Kubernetes, is written in Rust and has a code review document. The evidence collection model $\overline{m}$ contains the description of the execution flows to collect evidences on $\overline{ToC}$ to verify the property $\overline{p}$. For instance evidence collection model for artefacts dimension $\overline{m}_{art}$ =*{*(*Replica Manager=*Kubernetes, Get/Orchestrator), (*Replica Manager=*Kubernetes, Check-/Replicas), (*Replica Manager=*Kubernetes, Check/Zones)*}*, where Get/Orchestrator, Check/Replicas, Check/Zones are different evidence collection flows (i.e., probes) that first retrieve the orchestrator, then verifies the number of replicas and zones.*

We note that the Example 4.1.2 shows attributes that refers to deployment environment, however no specific evaluation is carried out on the environment apart from specific checks in $\overline{m}$ for configurations supporting the given property. More advanced verification at deployment level are presented in Section 4.3 The multi-factor scheme provides benefits for all the involved parties in the certification process as follows. The Service provider retrieves certificates better reflecting the behavior of its services [46, 47, 48]. The End user selects services according to more accurate certificates, supporting fully/informed and safe decisions. The CA improves the quality and in turn the trustworthiness of its certification scheme, providing higher accuracy with a marginal increase in overhead.

4.2 Certification of Composition

Service composition is bringing the dynamicity of the cloud to an unprecedented level. A service can be dynamically offer to a given client by combining already existing services into a new composed one. In this context, a certification scheme must consider the composition as a whole including all the phases from service selection to the non-functional evaluation of the new composed service.

Most of the solutions addressing composition of services present in literature focused on non-functional aware composition generation [49, 50, 51, 52, 53] or on QoS-aware composition monitoring and building [54, 55, 56, 57] These solutions are not fully suitable to be used in the context of certification due to

the lack of evidence and structured models that can be part of a trustworthy certification methodology. Most of the certification-specific approaches present in literature focused on service compositions require intensive involvement of CA and accredited lab in order to generate a composed certificate[58].

This collides with the extreme dynamicity of the compositions that often change run-time to address specific requirements (e.g., performance). There is a clear evolving trend of lightweight composition certification schemes supporting dynamic composition including run-time component substitution [59].

A composition certification process requires three fundamental building blocks: i) Composition modelling, driving the functional and non-functional building and evolution of the composition, ii) Component selection, allowing to select suitable component eventually at run-time in order to replace services and iii) Certification of Composition, generating certificate for the dynamically generated composition of services. In the following we describe each of these building blocks.

4.2.1 Composition Modelling

Traditionally two different type of composition of services are considered: i) orchestrations, where a central entity (the orchestrator), intermediate the communication among the component services, and ii) choreography, where services directly exchange messages in the framework of the composition. In the following we consider orchestration-based composition. Among the possible language to be used to model a composition of services (e.g., OWL-S[]), BPEL is one of the more largely used specifically for orchestration-based compositions. BPEL is an XML-based language that allows to define a business process by using a collection of services. Mores specifically it allows to i) specifying the order in which service operations are invoked, ii) the data to be exchanged at each step of the composition, and iii) the conditions under which a service is selected and integrated within the business process [60]. BPEL defines executable processes that consist of a set of activities (e.g., *invoke*, *receive*, and *reply*) combined using different structures to form a coherent system. BPEL engine in most of the cases, supports run-time service composition, where component services, that are listed in a registry, are dynamically selected and composed on the basis of functional requirements [**?**]. We model the BPEL service composition as a graph as follows.

Definition 4.2.1 (Composition graph) *A Composition graph G(V,E) is a direct acyclic graph with a root $v_r \in V$. G also has i) a vertex $v_i \in V_I \subseteq V$ for each service operation invocation, ii) two additional vertices $v_c, v_m \in V_\otimes \subset V$ for each alternative ($\otimes$) structure modeling the alternative execution (*choice*) of operations and the retrieval (*merge*) of the results, respectively, and iii) two additional vertices $v_f, v_j \in V_\oplus \subset V$ for each parallel ($\oplus$) structure modeling the contemporary execution (*fork*) of operations and the integration (*join*) of their results, respectively.*

We note that $\{v_r\} \cup V_I \cup V_\otimes \cup V_\oplus = V$, and v_c, v_m, v_f, and v_j model branching for alternative/parallel structures.

We also note that root vertex v_r represents the BPEL orchestrator, that is, the set of operations exposed in the interface by the process owner. For simplicity we consider the orchestrator as free of internal functionality to be certified and just as a means to trigger invocations.

4.2.2 Composition Annotation

Annotations have been proposed in the past to support selection of services to be integrated within a composition [50, 51, 53]. Each BPEL invocations can be annotated detailing non-functional property to be satisfied by the component services, and in turn detailing the propriety for which it has to be certified for. It can be seen as a labeling function $\lambda: V_I \rightarrow L_S$ that associates a set of non-functional requirements $\mathcal{R} \in L_S$ with each invocation $v \in V_I$. We formally define an annotated BPEL graph as follows.

Definition 4.2.2 (Annotated Composition graph) *An annotated Composition graph $G^\lambda(V, E, \lambda)$ of a given a Composition graph G(V,E), is a direct acyclic graph with a labeling function λ that assigns a label $\lambda(v)$ to each service operation invocation represented as vertex $v \in V_I$. $\lambda(v)$ corresponds to the non-functional property to be satisfied by the component service represented by v.*

We note that security annotation $\lambda(v)$ can be expressed using metadata format like XML having the scope of modeling requirements in a rich and structured form. For instance, in case the selection have to be done among certified services, it is possible to specify details on the requested property p and evidence ev.

Figure 4.1 shows an annotation expressed as XML fragment requesting a service certified for property *Confidentiality* of data *in transit* using SSL/TLS with a key length equal to $2048bit$. It also requires evidence composed of at least 200 test cases of category *Penetration*.

The Composition graph with annotations is at the bases for our *Composition Template* $\mathcal{T}^{com}$ with is driven by a non-functional property p a service owner wants to certify on its composite service. This property is declined into the annotations on each component services constituting the annotated graph in Definition 4.2.2 and the target ToC is the annotated graph $G^\lambda(V, E, \lambda)$.

Definition 4.2.3 (Composition CM Template $\mathcal{T}^{com}$) *A Composition CM Template $\mathcal{T}^{com}$ is 5-tuple of the form $\langle p, G^\lambda(V, E, \lambda), m, ev, l, \rangle$ where the property p is the property to be certified for the entire composition the annotated graph $G^\lambda(V, E, \lambda)$ is the compositional target of certification declining p into specific requirements for each component service, m^f and ev are the same as in traditional CM Template but contain component-specific and composition as a whole-specific evidence collection models and evidence respectively. Life cycle l is expressed in terms of evidence ev as usual.*

```
<annotation id="01" ref="Invocation">
   <property>
      <abs>Confidentiality</abs>
      <algo>SSL/TLS</algo>
      <key>2048 bit</key>
      <ctx>in transit</ctx>
   </property>
   <evidence>
      <category>Penetration</functionality>
      <attrs>
         <attr id="cardinality">200</attr>
      </attrs>
   </evidence>
</annotation>
```

Figure 4.1: An example of non-functional annotation for a given invocation in the BPEL

Similarly to CM Template the process of defining the Composition CM Template may involve the CA, specifically to carry out annotation on G^{λ} and an accredited lab to verify its validity.

4.2.3 Candidate Service Selection

In order to generate an executable composition showing a specific non-functional property, component services have to be selected and connected to the Composition Template $\mathcal{T}^{com}$ building the Compositional Instance $\mathcal{I}^{com}$. Many solutions exists in literature to select services based on their non-functional properties (e.g., based on SLA annotations [18, 61, 20] or based on certificates [62]).

The service selection can be considered as a function that takes as input an Composition Template $G^{\lambda}(V, E, \lambda)$ and different sets of candidate services, each one satisfying the functional properties of one service operation invocation, and returns as output an executable composition $G'(V', E)$ where every invocation $v \in V_I$ contains a service instance v', and every branching $v \in V_{\otimes} \cup V_{\oplus}$ is maintained as it is.

The executable composition $G'(V', E)$ is obtained by traversing the graph G^{λ} with a *depth-first* search. The algorithm starts from the root vertex v'_r of G' containing the instance of the orchestrator. Then for each vertex $v \in V_{\otimes} \cup V_{\oplus}$, a corresponding vertex $v' \in V'_{\otimes} \cup V'_{\oplus}$ is generated where each vertex $v' \in V'_I$ is an instance of a real service wsi, such that the wsi functionally compatible with the functional requirements in $G^{\lambda}(V, E, \lambda)$ and satisfies the annotation $\lambda(v)$.

More in details to select the candidate service for v' a two-step candidate selection approach is applied as follows.

1. *Match:* Consider a set S of candidate services s_i and $\lambda(v)$ as the non-functional requirements for the invocation at vertex v in the Composition Template. Assuming that functional matching is successful, non-functional requirements are considered by evaluating if s_i satisfies $\lambda(v)$, (e.g., s_i owns a certificate C_i satisfies $\lambda(v)$). The matching algorithm returns a set $S' \subseteq S$ of compatible services, which represent the possible

candidates for selection.

2. *Rank:* Compatible services $s_i \in S'$ are ranked in a partial or full order on the basis of their non-functional characteristics (e.g., on the bases of their certificates C_i) . The best ranked service is then selected and integrated in $v' \in V'_I$.

We note that the two steps matching and ranking depends on how the non-functional property of a given component service is expressed both in the annotation and as metadata associated to the component service.

In the case of certified component services, in the multi-factor scenario, matching may need to check all the factors detailed in the certificate (see Section 4.1) and the ranking may need to rank based on multiple factors [45]

The executable composition $G'(V', E)$ is used to build the Composition CM Instance $\mathcal{I}^{com}$ as follows.

Definition 4.2.4 (Composition CM Instance $\mathcal{I}^{com}$) *A Composition CM Instance $\mathcal{I}^{com}$ is a 5-tuple of the form $\langle \overline{p}, G'(V', E), \overline{m}, \overline{ev}, \overline{l} \rangle$ where $\overline{p}$ is the concrete property, $G'(V', E)$ is the executable composition constituting the concrete Target of Certification $\overline{ToC}$, $\overline{m}$ represents the executable evidence collection activities to be carried out on the composed service whose results (the evidence $\overline{ev}$) are evaluated following certificate life cycle $\overline{l}$.*

4.2.4 Composition Certification

Composition CM Instance $\mathcal{I}^{com}$, the service provider can either decide to certify the entire composition considering each component and the Composition Template as the model to compose them, or, in case the component services are already certified in a stand alone fashion, can re-use the certificates available for the component services to generate the certificate of the composition. More specifically:

- Composition Full-Certification: The Composition Template and Instance are used to set up a complete certification process as in Section **??**. More specifically the $\overline{m} \in \mathcal{I}^{com}$ represents the evidence collection flows targeting each component services are in traditional certification (i.e., testing a software component). The evidence $\overline{ev}$ collected represents results from the execution of the different flows in $\overline{m}$ on the real software components. This process generates a traditional certificate having evidence generating by the execution of the given $\mathcal{I}^{com}$.

- Composition Re-Certification: Each component services show a stand-alone certificate for which it was selected (see Section 4.2.3). The Composition Template can be used similarly to a CM Template for the composition and part of the CM Instance of each component services can be reused and/or re-executed in the framework of a complete certification process for the entire composition. Both $\overline{m} \in \mathcal{I}^{com}$ and $\overline{ev} \in \mathcal{I}^{com}$ points

to the relative $\overline{m}$ and $\overline{ev} \in \mathcal{I}$ of each component part of the composition. This process generates a certificate were evidence are not generated from scratch as for the Full-Certification but, the already available evidence at each component level are just re-verified if needed.

- Virtual Composition Certification: The core idea is to empower CA to virtually certify a composite service starting from the certificates of the component services. More specifically the $\overline{m} \in \mathcal{I}^{com}$ represents checks on the certificate of the component services while the evidence $\overline{ev}$ represents the results of such checks. It generates a composite service certificate that is "virtual", since it do not involve any real evaluation activities [59]. The trust in a virtual certificate is mainly based on the trust a customer has in the automatic process implemented by the accredited lab delegated by the CA and in the certificates held by the individual component services.

Example 4.2.1 (Virtual Certification) *Let us consider the Scenario in Section 3.2 which is made of a composition of four services. Let's assume that the services are already certified and selected based on their certificates in order to provide property (*Confidentiality, {`ctx` = in-transit*). More specifically two storing services: Artefacts Storage (s_2) and Compliance Controls Repository (s_3) both certified for Confidentiality at rest and Integrity at rest ($\mathcal{C}_{s_2}$, $\mathcal{C}_{s_3}$), the Execution Manager (s_4) certified for Confidentiality in transit ($\mathcal{C}_{s_4}$), and an orchestrator service, the Compliance Manager (s_1) certified for authenticity and confidentiality in transit ($\mathcal{C}_{s_1}$). Considering the above certificates and the composition structure, the composition can be awarded with the virtual certificate $\mathcal{C}_{ws_c}$ $\langle \mathcal{I}^{com}, \mathcal{T}^{com}, ws_c, \overline{ev}_r \rangle$ for property (*Confidentiality, {`ctx` = in-transit}*), where the $\mathcal{I}^{com}$ is based on the $\mathcal{I}$ of each component, as well as the the evidence $\overline{ev}_r$ is based on the composed virtual evidence. Let's now assume that the certificate $\mathcal{C}_{s_2}$ is suspended for property Confidentiality and still valid for property Integrity, then the entire virtual certification is suspended as well since Integrity is not needed as a property in per framework of the virtual certification process for Confidentiality.*

4.3 Certification of Deployment Environment

The deployment environment has a great impact on the property of the service executed on it. The most simple impact is on performance that a given powerful deployment can guarantee in comparison with a weaker one. The deployment can also offer security features to strength the deployed services or security weaknesses impacting the service security (e.g., encrypted file system vs. a plain one). It is therefore fundamental con consider the deployment environment properties in the framework of certification of services. Deployment environment can be partially involved as a factor in a multi-factor certification such as the one in Section4.1 where a factor can be the environment of the deploy. For instance in Example 4.1.2 Kubernetes deployment environment configurations are involved in the artefacts factor for certification of reliability for the given service. In

case of multi-factor certification, however, the target is the system/service and not the environment, therefore just some aspects relevant for the service under certification is considered and just for the scope of the certification. In this section we cover the certification of deployment environment as the sole target of the certification process independently for services that will be hosted on it. Certification of deployment environment shares a lot of commonalities with certification of Service Containers. In both the scenarios, the scope is to certify properties that can then later be inherited by a service executed within the container or in a specific environment. The certification activities are, in most of the solutions present in the literature, very similar to traditional service certification []. Most of the approaches presented in literature just re-apply service certification targeting the environmental infrastructure or the Web Service Containers []. More recently a different approaches were proposed having the objective of empowering the environment with hooks exposing metrics to make them certification-ready and to ease the integration into scenarios of service composition certifications [63, 64]. This approaches are becoming even much more interesting with the advent of edge solutions involving telco operators in service deployment and offerings [65].

4.3.1 Certification Modeling

Certification model for deployment environment is grounded on a different perspective then certification for services. First of all as a platform the deployment environment will host a number of services and its willingness of being certifiable is much more credible then a service per se. Its certification is also attractive from a business perspective and definitely open to the composition of certified services as described in Section4.2. The certification model therefore assumes a more collaborative behaviour by environment owner in providing means of collecting evidence, but on the other side given its multi-tenancy nature it is less worthy to have a plethora of specified probes deployed and executed in this shared environment. In addition modern deployment environment is increasingly based on telco facilities such as MEC edge services, which has a more restricted access to external inspections. The certification model is grounded on the assumption than the environment provides hooks offering relevant *Metrics* to be aggregated as *Rules*. The rules are meant to evaluate a specific aspect of the target environment. They are aggregated in terms of *Policies* defined by a CA each of them aimed to prove a specific behavior supporting a given *non-functional property*. Given the nature of the metrics and of the platform differently from the other certification processes the time window of measurement is considered to indicate the period of validity as well as to drive the aggregation of metrics.

The CM template for deployment environment $\mathcal{T}^{dep}$ is structured as follows.

Definition 4.3.1 (Deployment CM Template $\mathcal{T}^{dep}$) *A $\mathcal{T}^{dep}$ is 5-tuple of the form $\langle p,\ ToC,\ t,\ pol,ev,\rangle$ where the property p is the property to be certified, ToC is the target of certification expressed as the different mechanisms at*

environmental level supporting the property to be certified, t is the time interval for the evaluation and policies pol represents the generic policies defined in terms of rules to certify the property p. The collected evidence ev represents the expected outcomes of rules expressed in terms of metrics.

We note that life cycle is not specified since it is included into how the policy is specified in terms of rules. As for the other Certification processes the CM template $\mathcal{T}^{dep}$needs to be instantiated in order to the executable by the accredited lab. In this case it meas that rules and metrics that are part of the policies *pol* in the $\mathcal{T}^{dep}$need to be specified concretely on the real target.

Definition 4.3.2 (Deployment CM Instance $\mathcal{I}^{dep}$) *A $\mathcal{I}^{dep}$ is 5-tuple of the form $\langle \overline{p}, \overline{ToC}, t, \overline{pol}, \overline{ev}, \rangle$ where the property $\overline{p}$ is the concrete property to be certified, $\overline{ToC}$ is the concrete target of certification linking the concrete executable environmental level mechanisms supporting the property to be certified, t is the time interval for the evaluation and policies $\overline{pol}$ represents the concrete policies defined in terms of concrete rules exprressed in terms of real metrics aimed to certify the property $\overline{p}$. The collected evidence $\overline{ev}$ represents the concrete outcomes of rules expressed in terms of metrics.*

Let us consider a given $\mathcal{I}^{dep}$ expressing a non-functional property (*Confidentiality*, {`ctx` = *in-transit/between nodes*}) to be certified for a target environment $\overline{ToC}$ made of Information Centric Nnetwork nodes. The accredited lab implements the CA's policies in $\mathcal{T}^{dep}$as $\overline{pol}$ in $\mathcal{I}^{dep}$ in terms of rules aggregating metrics with the aim of collecting the evidence $\overline{ev}$ supporting the property. Two rules can be defined as follows $r_i(\mathbf{n}, \mathbf{t}) = m(\mathbf{n}, \mathbf{t}) \geq 2048$ where the metric m indicates the size of the RSA key used to encrypt all communications between the nodes in $\mathbf{n}$ in a given time interval $\mathbf{t}$. The second rule $r_j(\mathbf{n}, \mathbf{t})$ is true *iff* $\mathbf{n}$ uses valid certificate for message encryption in the time interval $\mathbf{t}$. Accredited lab evaluates the three rules included in the policy against a given set of nodes $\mathbf{n}$ and a time interval $\mathbf{t}$. If successful, it issues a certificate including the policy, the metrics, and the evidence obtained. Given three nodes $\mathbf{n} = \{a, b, c\}$ and a time frame $\mathbf{t} = [11.2, 33.5]$ as input for the verification of the policy instance $\overline{pol}$, if $\overline{pol}(\mathbf{n}, \mathbf{t}) = \text{true}$, the certificate is issued and the evidence $\overline{ev}$ is collected during the evaluation: $\overline{ev}_1(\{a\}, t) = 2048$, $\overline{ev}_1(\{b\}, t) = 2048$, $\overline{ev}_1(\{c\}, t) = 2048$, $\overline{ev}_2(\{a\}, t) = true$, $\overline{ev}_2(\{b\}, t) = true$ and $\overline{ev}_2(\{c\}, t) = true$.

Example 4.3.1 (Certification of Containerization) *Let us consider the Example 4.1.2 where Kubernetes is involved in one of the factor for the certification of property reliability. Let us then consider this Kubernetes cluster as the target ToC of a deployment environment certification for the same property (*Reliability*,* {`ctx` = Replication*). The scope of this certification is to verify that the ToC support reliability via replication. The relative $\mathcal{T}^{dep}$includes generic policies pol with rules requiring that the target Kubernetes cluster shows replication of nodes. It is instantiated in the $\overline{pol}$ of the CM Instance $\mathcal{I}^{dep}$ where rules are implemented and metrics collected. For instance $\overline{pol}$ includes a rule*

instance $r_1(Kubernates\ cluster, \mathbf{t})$ *that is true iff the Kubernates shows a working cluster in HA in the time interval of the evaluation* $\mathbf{t}$*. It means that metrics monitoring the availability of the different cluster nodes are provided and collected by the rule* r_1 *in the time interval* $\mathbf{t}$*. We note that other rules collecting configurations details are part of the given* $\overline{pol}$ *as well.*

We note that differently form Example 4.1.2, the certification activities are specifically focused on the Kubernetes cluster and not on the service deployed on it. We also note that some of the evaluation activities in Example 4.1.2 can be also executed at deployment level as soon as a metrics are made available by the infrastructure owner.

4.4 Certification of The Development Process

Certification of Development Process aims to extend the concept of *continuous certification* with the scope of covering as much as possible the software life cycle from solution design, its coding and continuous updating while in operation. Some traditional approaches focused on the certification of some aspects of the development process only [66, 67], just few took the specific system under development into account [68] and even fewer addresses modern development processes [69]. In the context of certification of development process, the certification activities need to target all the software artifacts (including the intermediate ones) and bind collected evidence to all development stages, linking evidence back to certification requirements. This process "shift to left" the non-functional activities that usually has been thought as a "bolt-on" process after the provisioning of the application. More in detail, this notion of continuous certification introduces new requirements on certification schemes that are summarized as follows.

- **Development process modeling:** the development process must be part of the certification model. This modelling is under the responsibility of the CA.
- **Stage-by-stage evaluation:** the certification model must describe both requirements and the corresponding controls used to collect relevant evidence at each development stage.
- **Trustworthy inspection:** the development process must allow trustworthy execution of controls.
- **Continuous verification:** hooks in the development process must be provided to support controls in continuously collecting evidence.

The certification of development process includes both certification of final target system and the certification of the deployment environment where the system will be deployed into a complete and continuous certification process.

Although continuous certification (and corresponding requirements) is applicable to any development process, it provides the best advantages in agile

processes, where Continuous Integration (CI) and Continuous Delivery (CD) make the development process and the software it releases a unique conceptual entity. In the following, we then focus on certification of services developed following a DevOps process, defining a new certification scheme for DevOps. DevOps in fact provides full control and automation of the development steps, and is a natural candidate for supporting continuous certification.

An overview of the certification activities done at each DevOps stage is sketched in the following.

- Stage *Plan* designs the application. It also specifies security requirements and metrics to measure performance and quality of service of the application itself.
- Stage *Create* generates the application code, statically verifies it for security reasons, and builds it after the source code is committed.
- Stage *Verify* provides features for functional and non-functional testing of the software.
- Stage *Package* prepares the software artifacts for application deployment, and the corresponding security verification on packaging dependencies.
- Stage *Release* marks the difference between staging and operation pipelines targeting private or public deployments, respectively.
- Stage *Configure* configures the deployment environment, and includes infrastructure-level and container-level security verification.
- Stage *Monitoring* continuously monitors the application to calculate those metrics needed for continuous security verification. We note that stages Release, Configure, and Monitor consider both staging and operation environments.

4.4.1 Certification Modeling

Similarly to the traditional Certification schemes, development process certification is modeled in a CM Template ($\mathcal{T}^{dev}$) that incarnates the need of verifying the adherence of the whole development process to relevant policies and requirements. More specifically the CM Template $\mathcal{T}^{dev}$is focused on triggering specific verification activities C at each stage of the development process. The target T of these verification activities can be software components like in the case of non functional mechanisms θ, or configuration files and even planning documentation files.

The Development Certification Model Template $\mathcal{T}^{dev}$ is structured as follows.

Definition 4.4.1 (Development CM Template $\mathcal{T}^{dev}$) *A $\mathcal{T}^{dev}$ is a 4-tuples of the form <$phase_i$, T_i^s, C_i, Υ_i>, where $phase_i$ refers to the i-th DevOps phase*

$phase_1$="Plan" T_1^s={$T_{1,1}^s$="Solution design"} C_1={c_1="Verify requirements document"} Υ_1= =({$c_{1,1}$,T_1^s}, success)
$phase_2$="Create" T_2= {$T_{2,1}^s$="Requirements document", $T_{2,2}^s$="Code repository"} C_2= {$c_{2,1}$="Check requirements links", $c_{2,2}$="Code analysis"} Υ_2= ($c_{2,1}$,$T_{2,1}^s$}, success) $\wedge$ ({$c_{2,2}$, $T_{2,2}^s$, success)
$phase_3$="Verify" T_3^s={ $T_{3,1}^s$="Latest release"} C_3={$c_{3,1}$= $\mathcal{T}$i} Υ_3= ({$c_{3,1}$, T_3^s}, success)
$phase_4$="Package" T_4^s={$T_{4,1}^s$="Dependencies"} C_4={$c_{4,1}$="Dependencies vulnerability checks"} Υ_4= ({$c_{4,1}$ T_4^s}, success)
$phase_5$="Release" T_5={$T_{5,1}^s$="Deployment environment"} C_5={$c_{5,1}$="Release procedure checks"} Υ_5=({$c_{5,1}$, T_5^s}, success)
$phase_6$="Configure" T_6^s={$T_{6,1}^s$="Deployment Infrastructure"} C_6={$c_{6,1}$=$\mathcal{T}_1^{dep}$, $c_{6.2}$=$\mathcal{T}_2^{dep}$} Υ_6=({$c_{6,1}$, T_6^s}, success) $\vee$ ({$c_{6,2}$, T_6^s}, success)
$phase_7$="Monitor" T_7^s={$T_{7,1}^s$="Application"} C_7={$c_{7,1}$="Vulnerability scan", $c_{7,2}$="Pen test", $c_{7,3}$="Execution trace checks"} Υ_7=({$c_{7,1}$, T_7^s}, success) $\wedge$ ({$c_{7,2}$, T_7^s}, success) $\wedge$ ({$c_{7,3}$, T_7^s}, success)

Figure 4.2: An excerpt of the Development Certification Model Template $\mathcal{T}^{dev}$ for the DevOps process of the service in Example4.1.2.

triggering the execution of certification activities in C_i, $T_i^s \subseteq T$ is an artifact of the system under certification (e.g., source code, executable, configuration files), $C_i \rightarrow \{c_{i,1},\ldots,c_{i,n}\}$ is a list of generic control types suitable for system certification, and Υ_i defines the guards that need to be satisfied for triggering a phase transition.

We note that each C_i specifies a class of controls that points to a set of concrete controls $\{c_{i,1},\ldots,c_{i,n}\}$ to be used for system checks. For instance, class *web vulnerability* includes all the controls that provide web vulnerability assessment functionalities. We also note that guards Υ_i are defined as a boolean conditions of the form $op(\{c,T^s\},EO)$, where op is an operator in $\{=,\leq,\geq,<,>\}$, c is a control, T^s is the target of the control, and EO is the expected output of the execution of control c on target T^s. Guards express conditions to be satisfied by the specific step in order to proceed with subsequent ones. We note that if a guard is not satisfied, the certification process ended. In the case of failures correction activities might be requested. We also note that, for some targets T^s, the c are the traditional ones used in solution such as the one in [**?**].

Figure 4.2 shows an extract of a Development Certification Model Template $\mathcal{T}^{dev}$ of the DevOps Process associated with the production of the service in

Example4.1.2.
$phase_1$ refers to stage Plan. C_1 contains a single manual control $c_{1,1}$ ensuring that the requirements document exists and has the format requested to drive the remaining of the process (Υ_1).
$phase_2$ refers to stage Create. C_2 contains two types of controls: *i)* $c_{2,1}$ verifies whether the requirements document in stage Plan ($T^s_{2,1}$) has been manually annotated to link requirements to the source code components they regulate; *ii)* $c_{2,2}$ statically verifies the source code in the code repository ($T^s_{2,2}$) to identify security issues. In case both $c_{2,1}$ and $c_{2,2}$ succeed (Υ_2), the following stage is executed.
$phase_3$ refers to stage Verify. C_3 contains a single control $c_{3,1}$. This control can refers to a traditional CM Template $\mathcal{T}$i focused on ad hoc security testing on the latest version of the build release ($T^s_{3,1}$).
$phase_4$ refers to stage Package. C_4 contains a single control $c_{4,1}$ executing dependency vulnerability checks on the additional packages ($T^s_{4,1}$), if any, required for application packaging.
$phase_5$ refers to stage Release. C_5 contains a single control $c_{5,1}$ verifying the correctness of the release procedure.
$phase_6$ refers to stage Configure. C_6 refers to two deployment CM Templates $\mathcal{T}_i^{dep}$ both having as target to deployment infrastructure ($T^s_{6,1}$): *i)* $c_{6,1}$ defined in $\mathcal{T}_1^{dep}$ and focused on compliance to Center for Internet Security benchmark; *ii)* $c_{6,2}$ defined in $\mathcal{T}_2^{dep}$ and focused on checking the absence of exploitable vulnerabilities collecting metrics based on vulnerability scan and penetration testing. We note that the activities listed above are based on probes periodically executed on the infrastructure the results of which are made available as metrics.
$phase_7$ refers to stage Monitor. C_7 contains three types of controls both having as target to system application instead of the infrastructure ($T^s_{7,1}$): *i)* $c_{7,1}$ performs a vulnerability scan; *ii)* c_2 performs a penetration testing; and *iii)* c_3 monitors the executions of the target system when deployed in the operation environment [**?**].

Phases $phase_5$, $phase_6$, $phase_7$, that refers to DevOps stages Release, Configure, and Monitoring, respectively, needs to be applied both in staging and operation environments. The development process certification is a process strongly coupled with the pipelines used for software production. It is not just applied to verifies the final deployed in operation but to continuously verifies all artifacts produced even in staging.

We also note that, the Development Certification Model Template $\mathcal{T}^{dev}$ is a composite model where certification activities are independently specified for each DevOps stage. Differently form the other certification models, given the composition nature of the template, a certification model istance is notm defined as such. The certification activities are mapped to the DevOps process by multiple instantiation functions λ_i generating specific instances of each steps. At a logical level, when the i-th DevOps stage is triggered, the corresponding λ_i is executed and the certification activities in $\mathcal{T}^{dev}$ for that stage is instantiated on the relevant system artifact T^s_i (e.g., code, package). We note that if the

verification of a give stage is described itself as a specific CM Template (e.g., at the verify and configure stages) the relative instantiation function λ_i generates and executes the corresponding and specific CM Instance.

The result of the execution of all the instantiation functions is an instance $\mathcal{I}^{dev}$ of the Development Certification Model Template $\mathcal{T}^{dev}$ which drives real certification activities along the DevOps pipeline.

We recall that phase Plan in $\mathcal{T}^{dev}$ requires manual activities; therefore, the instantiation function executes manual checks. For performance reasons, the execution of instantiation functions λ can be pre-computed for each development round.

Chapter 5

Part 4: Beyond Cloud Service Certification

Cloud computing has been the seed for multiple technological revolutions in the last two decades, which built on device miniaturization and sensors, and resulted in data-centric environments. Modern distributed systems are deployed on cloud-edge continuum enabled by 5G networks, are centered around machine learning and artificial intelligence, and fundamentally changed the behavior of traditional (cloud-based) distributed systems. Certification schemes have to keep the pace of this continuous system evolution, which opens the door to multiple challenges. This chapter analyses the need of redesigning current certification schemes to address the requirements introduced by modern distributed systems.

5.1 Certification of Cloud-Edge Distributed Systems

Modern systems are increasingly moving from a centralized architecture where computations are done at the core of the network, as for instance in a cloud infrastructure, to a decentralized architecture where part of the computations are executed at the edge of the network, near the data collection points. This scenario is often referred to as cloud-edge continuum, where complex computation pipelines are deployed on the continuum and connected to the data sources that are often localized in the Internet of Things (Figure 5.1).

Internet of Things (IoT) can be defined as "*the networked interconnection of everyday objects, equipped with ubiquitous intelligence*" [70]. It is composed of physical devices from minuscule sensors to big components that are connected by fast wired and mobile networks, and contributes to the implementation of smart services delivered at the edge. IoT is a fundamental building block of the cloud-edge continuum connecting billions of devices sensing the physical environment

Figure 5.1: TODO - Cloud-Edge Continuum.

through high-throughput and ultra-low latency (5G) communications. This architecture pushed forward a data-driven ecosystem, where a huge amount of data are created, collected, consumed, and stored, and cannot be effectively handled using traditional cloud infrastructures.[1] Edge nodes are then used to pre-process data at the edge and forward them to the cloud where cloud-based applications are built on top of processing services.

This new ecosystem comes with important advantages in terms of applications and added value for its users, making their world smarter and simpler. However, the price we pay for such advantages is an increasing difficulty in managing non-functional aspects including security, privacy and trustworthiness, which are must-have requirements in scenarios where the users are yet another system component, and are surrounded and interact with a multitude of systems/sensors.

5.1.1 Challenges

Modern systems pose strict requirements on the need of redesigning and rethinking current certification schemes for the cloud (Chapter 3.3), driven by a new set of challenges as follows.

- *Verification of heterogeneous and complex systems with unknown boundaries.* Modern distributed systems mix heterogeneous technologies in a multi-layer infrastructure, where a multitude of smart devices are connected through fast and reliable networks (e.g., 5G network). The certifi-

[1] Statista `https://www.statista.com/statistics/871513/worldwide-data-created/` claimed that the total amount of data created, captured, copied, and consumed globally reached 64.2 zettabytes in 2020, expecting to grow to more than 180 zettabytes by 2025, a number that is also shared by IDC (https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/?sh=2404c2385459) expecting to grow to 175 zettabytes by 2025

cation of these systems carries multiple challenges mostly connected to the variability of the environment and the intrinsic untrustworthiness of smart devices. More in detail, the reference environment quickly evolves over time in terms of participating entities and topology, resulting in an environment with unknown boundaries and diverse technologies. Also, smart devices are usually resource constrained, physically accessible, and protected with lower standards than the traditional cloud computing nodes. This breaks two of the assumptions at the basis of traditional cloud certification schemes: *i)* the ability of precisely defining the boundaries of the target of certification; *ii)* the trustworthiness of the target of certification as discussed in the following challenge.

- *Management of untrustworthy data and providers.* Traditional certification schemes build on the assumption of being able to collect trustworthy data from trustworthy/known providers. Modern systems built on smart devices cannot guarantee the above assumptions, since data are continuously collected by a multitude of devices, which are intrinsically unreliable and under the control of many untrusted providers. A novel certification scheme must depart from any assumptions on data/provider trustworthiness and consider a zero-trust environment.

- *Certification of resource constrained devices.* Modern certification schemes must be designed to have a limited impact on the target of certification. The latter, being composed on many resource-constrained devices, cannot be certified in deep with the risk of exhausting system resources. Evidence collection techniques (e.g., testing, monitoring) must balance between the completeness of retrieved evidence and the impact on the target components. Full certification can come at the price of being harmful to the system under certification, and must then be relaxed to limit the impact on its functioning.

- *Behavioral testing vs component testing.* As discussed in Chapter 3 traditional certification schemes are based on the detailed modeling of the target of certification and its components, which are monitored and tested both independently and in composition. Modern certification schemes must depart from this assumption and consider behavioral testing as a possible approach to certification. Behavioral testing consists in the modeling of the expected system behavior that is then matched against the observed system behavior [71].

5.1.2 Machine Learning for Cloud-Edge Certification

A certification scheme for the cloud-edge continuum must address three main requirements addressing the challenges in Section 5.1.1: *i)* a novel certification model capturing the behavior of a complex system in the cloud-edge continuum, *ii)* a distributed evidence collection process with minimum overhead on the target system and its resource-constrained devices, *iii)* a transparent and

Figure 5.2: TODO - New certificate life cycle.

continuous certification process that updates its model and corresponding certificates according to system changes.

As discussed in Section 3.3, traditional certification models are composed of three main parts specifying the property to be certified, the target of certification, the evidence collection model. The certification model represents the starting point of a novel certification process for the cloud-edge continuum. In particular, upon statically certifying a cloud-edge system in a controlled environment, a machine-learning based certification process should start to keep the validity of the certificate in operation and across system changes. The inherent format of a cloud-edge environment, with no fixed topology and resource-constrained devices, in fact, require new ways of modeling non-functional properties, as well new ways of modeling the target of certification and collecting evidence to address the challenges in Section 5.1.1. Non-functional properties must be redesigned to model requirements in multiple dimensions of interest (e.g., artifacts, development, evaluation) [72] and adapt to the system changes. Machine-learning techniques can be used to model the behavior of system under certification and drive evidence collection, managing the entire certificate life cycle. In particular, a continuous certification process can be defined, where the target of certification and the evidence collection model flows in a single behavioral component as follows:

1. Upon certifying the target of certification in a controlled environment, a model of its behavior is built using machine learning, with no requirements on the topology of the target or static assumptions on its composition. The behavioral model is built before the system is put in operation and becomes the cornerstone for certificate maintenance and continuous verification.

2. The behavioral model is then continuously checked against the model de-

rived by the system in operation. As soon as a disalignment between the original behavioral model and current behavioral model is detected (anomaly detection), an adaptation process is raised.

3. The adaptation process retrieves the differences between the two models and executes corrective actions that result in the *i)* confirmation of the original certificate, *ii)* refinement of the original certificate in a new one modeling the current status of the system, *iii)* revocation of the certification and triggering of re-certification from scratch.

In summary, service certification consists in the release of a certificate in a controlled environment using traditional certification schemes, followed by the modelling of the expected behavior of the system as whole and the continuous monitoring of certificate validity through an anomaly detector spotting any divergence from the expected behavior.

5.2 Certification of Machine Learning-Based Systems

The new shape of modern distributed systems, increasingly composed of micro- and nano-services orchestrated at run time to provide specific functionalities, pose some additional requirements on their governance. Traditional software systems are driven by deterministic algorithm, while modern systems are increasingly built on ML models. ML models are at the core of modern systems and drive their behavior, that is, they reason on data to calculate a solution to individual instances of a problem [73].

ML models are deeply changing system design and development, as well as their monitoring and verification. They are treated as black boxes, thus making automated decisions based on inference unpredictable and increasing safety and security risks on the final users. This scenario makes verification of system non-functional properties a major challenge by itself, and requires yet another call for new certification schemes that accomplish the peculiarities of machine learning. Lack of trustworthiness is in fact one of the main factors hampering ML-based system adoption in critical domains. In support of this view, Yoshioka and Ishikawa [74] conducted a study on several industry experts, which claimed that the assessment of non-functional requirements is among the most complex stages of a ML-based system development. ML models are in fact difficult to explain and monitor, thus impeding the adoption of assurance approaches including certification [75].

In this context, many new guidelines and regulations are defined/under definition concerning ML/AI trustworthiness, ethics, and risk. For instance, high-Level Expert Group on AI of the European Union presented the *Ethics Guidelines for Trustworthy Artificial Intelligence*[2] claiming that: "*As it cannot be expected that everyone is able to fully understand the workings and effects of*

[2]https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

AI systems, consideration can be given to organisations that can attest to the broader public that an AI system is transparent, accountable and fair. These certifications would apply standards developed for different application domains and AI techniques, appropriately aligned with the industrial and societal standards of different contexts." In 2021, the European commission then released the AI-act[3] claiming compliance and risk management as fundamental requirements for high-risk AI systems. These work agrees on the need of solutions for evaluating the non-functional shape of ML/AI systems, including certification, the focus of this book.

5.2.1 Challenges

Modern systems based on machine learning and artificial intelligence pose strict requirements on the redesign of current certification schemes. In particular, new schemes must be defined to support the certification of machine learning models, as follows.

- *Non-functional properties definition.* The non-functional verification of ML-based systems make current definition of non-functional properties (see Section 3.3) unusable. Attribute-based properties do not fit non-deterministic systems whose behavior is not known a priori. They must be replaced with properties that provides either a statistical, probabilistic or behavioral-based definition.

- *ML models are treated as black boxes.* The certification of ML based-service requires the ability to verify the ML black box. The emerging domain of ML verification uses behavioral monitoring techniques based on statistical testing to dynamically check properties (besides accuracy and fairness) of black-box ML models in operation. Monitors can be used to generate certificates for software applications, and composed to obtain dynamic virtual certificates for ensemble ML models. This however requires a proper definition of the entire certification chain, which must depart from the verification of deterministic systems based on traditional testing and consider the variability in behavior introduced by training data, configuration, and process.

- *Adaptive behavior.* ML-based systems are often unable to adapt their behavior to changing conditions, especially when non-functional aspects are considered. Degraded behavior is mostly considered in terms of accuracy and precision of the ML-based system, while no adaptation is possible when non-functional properties like fairness are considered. Also, it is usually not possible to adapt the ML-based behavior to contextual information as for instance the battery remaining on the specific devices executing the ML model. New certification scheme must be extended to support adaptive behavior of ML-based systems.

[3]https://artificialintelligenceact.eu/the-act/,https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai

- *Statistical testing.* As discussed in Section 5.1.1, traditional certification schemes are based on the detailed modeling of the target of certification and its components, which are verified both independently and in composition. This approach does not fit well ML-based systems, which require statistical testing and monitoring of ML models' behavior at inference time to efficiently check that the desired behavioral properties are achieved [76].

- *Evaluation of ML pipelines.* Similarly to traditional cloud-based certification, current systems are often built as a composite service, often referred to as machine learning pipelines. Certification of ML pipelines is inherently multilayer and multidimensional. First of all, each service in the pipeline must be certified within its deployment infrastructure. Afterwards, the certified services in the machine learning pipeline must be certified as a composite service verifying the orchestration of their functionalities.

5.2.2 Certification of ML-Based Systems

To address the above ML certification challenges it is necessary to reshape traditional certification schemes by revisiting the three main building blocks of the certification model in Section 3.3: non-functional properties, target of certification, evidence.

Non-Functional Properties. ML certification requires the definition of ML specific non-functional properties that redesign the way in which properties are defined and evaluate. Statistical and behavioral-based properties are needed to model the peculiarities of ML models. Some taxonomies have been already described (e.g., [76] and [4]) preliminary defining some properties as follows:

- *Transparency*: the feasibility of ex-post interpretation. It can further be specialized in *Explainability* [77] and *Interpretability* [78].

- *Fairness*: the absence of discrimination, that is, prejudice against an individual or a group based on the value of specific features [79].

- *Legality*: the compliance to laws and regulation, including privacy, quality, fundamental human rights, fairness, civil and criminal liability [80].

- *Maintainability*: the ability of going through complete or partial model retraining [81].

- *Modularity*: the ability to manage large or highly complex systems[82].

- *Performance*: the performance of a computation in terms of accuracy, precision/recall, Area Under the ROC curve.

[4] `https://www.enisa.europa.eu/publications/artificial-intelligence-cybersecurity-challenges?v2=1`

- *Privacy*: the requirement of deleting information about training set and those inferred from the models' output.
- *Reliability*: the ability to operate with no failures and guarantee a level of performance when operating in normal conditions [83].
- *Safety*: "*the expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered*"[84].
- *Scalability*: the ability to adapt to environmental changes and guarantee a sufficient level of performance.
- *Robustness*: the robustness against targeted or untargeted attacks. It includes the ability to protect data, models, hyperparameters, weights and coefficients used by models, and proprietary algorithms.
- *Usability*: the effort required by users to learn the software, prepare input data and interpret the results.

Target of Certification. It is natively multi-layer and includes three main factors to be considered: *i) data* including the description and metadata of the training, test, and validation sets. For instance, it can evalutes the granularity, sparsity, to name but a few; *ii) process* describing the details of the training process such as the preprocessing applied to data, the type of learning that is implemented, any other requirement on the model training; *iii) model* describing the model and its behavior.

Evidence collection model. It specifies all activities needed to collect evidence on a specific factor of the target of certification. It ranges from the verification of the correct behavior of the model, to the analysis of the possible data leak caused by observing the model in operation.

Chapter 6

Conclusions and Open Issues

Summary

Bibliography

[1] E. Damiani, C. A. Ardagna, and N. El Ioini, *Open source systems security certification*. Springer Science & Business Media, 2008.

[2] G. White, E. Fisch, and U. Pooch, "Government-based security standards," *Information Systems Security*, vol. 6, no. 3, pp. 9–19, 1997.

[3] S. B. Lipner, "The birth and death of the orange book," *IEEE Annals of the History of Computing*, vol. 37, no. 2, pp. 19–31, 2015.

[4] M. Gehrke, A. Pfitzmann, and K. Rannenberg, "Information technology security evaluation criteria (itsec)-a contribution to vulnerability?" in *IFIP Congress (2)*. Citeseer, 1992, pp. 579–587.

[5] E. M. Bacic, "The canadian trusted computer product evaluation criteria," in *[1990] Proceedings of the Sixth Annual Computer Security Applications Conference*. IEEE, 1990, pp. 188–196.

[6] P. Stephanow and N. Fallenbeck, "Towards Continuous Certification of Infrastructure-as-a-Service Using Low-Level Metrics," in *Proc. of IEEE UIC-ATC-ScalCom*, Beijing, China, August 2015.

[7] P. Stephanow, G. Srivastava, and J. Schütte, "Test-Based Cloud Service Certification of Opportunistic Providers," in *Proc. of IEEE CLOUD 2016*, San Francisco, CA, USA, June, July 2016.

[8] I. Kunz and P. Stephanow, "A Process Model to Support Continuous Certification of Cloud Services," in *Proc. of IEEE AINA 2017*, Taipei, Taiwan, March 2017.

[9] M. Anisetti, C. A. Ardagna, E. Damiani, and F. Gaudenzi, "A semi-automatic and trustworthy scheme for continuous cloud service certification," *IEEE TSC*, vol. 13, no. 1, 2020.

[10] S. Lins, S. Schneider, J. Szefer, S. Ibraheem, and A. Ali, "Designing Monitoring Systems for Continuous Certification of Cloud Services: Deriving Meta-requirements and Design Guidelines," *Comm. of the Association for Information Systems*, vol. 44, 2019.

[11] M. Alhamad, T. Dillon, and E. Chang, "SLA-Based Trust Model for Cloud Computing," in *Proc. of NBIS 2010*, Takayama, Japan, September 2010.

[12] Y. Du, X. Wang, L. Ai, and X. Li, "Dynamic Selection of Services under Temporal Constraints in Cloud Computing," in *Proc. of ICEBE 2012*, Hangzhou, China, September 2012.

[13] R. Shaikh and M. Sasikumar, "Trust Model for Measuring Security Strength of Cloud Computing Service," *Procedia Computer Science*, vol. 45, 2015.

[14] M. Eisa, M. Younas, K. Basu, and H. Zhu, "Trends and Directions in Cloud Service Selection," in *Proc. of IEEE SOSE 2016*, Oxford, UK, March, April 2016.

[15] F. Jrad, J. Tao, A. Streit, R. Knapper, and C. Flath, "A utility/based approach for customised cloud service selection," *International Journal of Computational Science and Engineering*, vol. 10, 2015.

[16] S. Ding, Z. Wang, D. Wu, and D. L. Olson, "Utilizing customer satisfaction in ranking prediction for personalized cloud service selection," *Decision Support Systems*, vol. 93, 2017.

[17] T. Halabi and M. Bellaiche, "A broker-based framework for standardization and management of Cloud Security-SLAs," *Computers & Security*, vol. 75, 2018.

[18] C. Redl, I. Breskovic, I. Brandic, and S. Dustdar, "Automatic SLA Matching and Provider Selection in Grid and Cloud Computing Markets," in *Proc. of ACM/IEEE Grid 2012*, Beijing, China, September 2012.

[19] A. Taha, R. Trapero, J. Luna, and N. Suri, "A Framework for Ranking Cloud Security Services," in *Proc. of IEEE SCC 2017*, Honolulu, HI, USA, September 2017.

[20] A. Taha, S. Manzoor, and N. Suri, "SLA-Based Service Selection for Multi-Cloud Environments," in *Proc. of IEEE EDGE 2017*, Honolulu, HI, USA, September 2017.

[21] K. J. Modi and S. Garg, "A QoS-based approach for cloud-service matchmaking, selection and composition using the Semantic Web," *Journal of Systems and Information Technology*, vol. 21, no. 1, Jan 2019.

[22] A. B. de Oliveira Dantas, F. H. de Carvalho Junior, and L. S. Barbosa, "A component-based framework for certification of components in a cloud of HPC services," *Science of Computer Programming*, vol. 191, 2020.

[23] D. Cotroneo, L. De Simone, and R. Natella, "Dependability Certification Guidelines for NFVIs through Fault Injection," in *Proc. of IEEE ISSREW 2018*, Memphis, TN, USA, October 2018.

[24] F. Nawab, "WedgeChain: A Trusted Edge-Cloud Store With Asynchronous (Lazy) Trust," in *Proc. of IEEE ICDE 2021*, Chania, Greece, April 2021.

[25] A. J. Muñoz-Gallego and J. Lopez, "A Security Pattern for Cloud service certification," in *Proc. of SugarLoaf PLoP 2018*, Valparaiso, Chile, November 2018.

[26] A. J. H. Simons and R. Lefticaru, "A verified and optimized Stream X-Machine testing method, with application to cloud service certification," *Software Testing, Verification and Reliability*, vol. 30, no. 3, 2020.

[27] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, 2018.

[28] S. Jahan, I. Riley, C. Walter, R. F. Gamble, M. Pasco, P. K. McKinley, and B. H. Cheng, "MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases," *Future Generation Computer Systems*, vol. 109, 2020.

[29] M. Anisetti, C. A. Ardagna, E. Damiani, N. El Ioini, and F. Gaudenzi, "Modeling time, probability, and configuration constraints for continuous cloud service certification," *Computers & Security*, vol. 72, pp. 234–254, 2018.

[30] R. Focardi, R. Gorrieri, and F. Martinelli, "Classification of security properties (Part II: Network security)," in *Foundations of Security Analysis and Design II - Tutorial Lectures*, R. Focardi and R. Gorrieri, Eds. Springer Berlin / Heidelberg, 2004.

[31] C. Irvine and T. Levin, "Toward a taxonomy and costing method for security services," in *Proc. of the 15th Annual Conference on Computer Security Applications (ACSAC 1999)*, Phoenix, AZ, USA, December 1999.

[32] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering, vol. 5.* Springer, Heidelberg, 2000.

[33] L. Chung and B. Nixon, "Dealing with non-functional requirements: Three experimental studies of a process-oriented approach," in *Proc. of the 17th International Conference on Software Engineering (ICSE 1995)*, Seattle, WA, USA, April 1995.

[34] L. Chung and J. Leite, "Conceptual modeling: Foundations and applications," A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. On Non-Functional Requirements in Software Engineering, pp. 363–379.

[35] M. Anisetti, C. Ardagna, E. Damiani, and F. Saonara, "A Test-based Security Certification Scheme for Web Services," *ACM Transactions on the Web (TWEB)*, vol. 7, no. 2, May 2013.

[36] M. Anisetti, C. A. Ardagna, E. Damiani, A. Maña, G. Spanoudakis, L. Pino, and H. Koshutanski, "Security certification for the cloud: The cumulus approach," *Guide to Security Assurance for Cloud Computing*, pp. 111–137, 2015.

[37] M. Anisetti, C. Ardagna, and E. Damiani, "Certifying security and privacy properties in the internet of services," in *Trustworthy Internet*, N. B.-M. L. Salgarelli, G. Bianchi, Ed. Springer, 2011.

[38] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "On a formal framework for security properties," *Computer Standards & Interfaces*, vol. 27, no. 5, pp. 457–466, 2005.

[39] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, "From security to assurance in the cloud: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–50, 2015.

[40] M. Anisetti, C. A. Ardagna, F. Gaudenzi, and E. Damiani, "A certification framework for cloud-based services," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 440–447.

[41] D. Herrmann, *Using the Common Criteria for IT security evaluation*. Auerbach Publications, 2002.

[42] M. Anisetti, C. A. Ardagna, and E. Damiani, "A low-cost security certification scheme for evolving services," in *2012 IEEE 19th International Conference on Web Services*. IEEE, 2012, pp. 122–129.

[43] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, 2004.

[44] V. Lotz, "Cybersecurity Certification for Agile and Dynamic Software Systems - a Process-Based Approach," in *Proc. of EuroS&PW*, Genoa, Italy, September 2020.

[45] M. Anisetti, C. A. Ardagna, and N. Bena, "Multi-dimensional certification of modern distributed systems," *IEEE Transactions on Services Computing*, 2022.

[46] H. Teigeler, S. Lins, and A. Sunyaev, "Drivers vs. Inhibitors - What Clinches Continuous Service Certification Adoption by Cloud Service Providers?" in *Proc. of HICSS 2018*, Waikoloa, HI, USA, January 2018.

[47] J. Lansing, A. Benlian, and A. Sunyaev, ""Unblackboxing" Decision Makers' Interpretations of IS Certifications in the Context of Cloud Service Certifications," *Journal of the Association for Information Systems*, vol. 19, 2018.

[48] J. Prüfer, "Trusting privacy in the cloud," *Information Economics and Policy*, vol. 45, 2018.

[49] Y. Bai, Y. Zhang, Y. Zhou, and L. T. Yang, "A non-functional property based service selection and service verification model," in *Proc. of UIC 2011*, Banff, Canada, September 2011.

[50] K. M. Khan, A. Erradi, S. Alhazbi, and J. Han, "Security oriented service composition: A framework," in *Proc. of IIT 2012*, Al Ain, UAE, March 2012.

[51] A.R.R. Souza et al., "Incorporating security requirements into service composition: From modelling to execution," in *Proc. of the ICSOC-ServiceWave 2009*, Stockholm, Sweden, November 2009.

[52] L. Pino and G. Spanoudakis, "Finding secure compositions of software services: Towards a pattern based approach," in *Proc. of NTMS 2012*, Istanbul, Turkey, May 2012.

[53] H. D. Vo, D. C. Phung, V. Q. Dung, and V.-H. Nguyen, "Securing data in composite web services," in *Proc. of KSE 2012*, Danang, Vietnam, August 2012.

[54] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient web service composition with end-to-end qos constraints," *ACM TWEB*, vol. 6, no. 2, pp. 1–31, June 2012.

[55] Y. Chen, J. Huang, C. Lin, and J. Hu, "A partial selection methodology for efficient qos-aware service composition," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 384–397, 2015.

[56] S. Deng, H. Wu, D. Hu, and J. Zhao, "Service selection for composition with qos correlations," *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 291–303, 2016.

[57] H. Zheng, J. Yang, and W. Zhao, "Probabilistic qos aggregations for service composition," *ACM Transactions on the Web*, vol. 10, no. 2, pp. 12:1–12:36, May 2016.

[58] M. Anisetti, C. Ardagna, and E. Damiani, "Security certification of composite services: A test-based approach," in *Proc. of the 20th IEEE International Conference on Web Services (ICWS 2013)*, San Francisco, CA, USA, June–July 2013.

[59] M. Anisetti, C. Ardagna, E. Damiani, and G. Polegri, "Test-based security certification of composite services," *ACM Transactions on the Web (TWEB)*, vol. 13, no. 1, pp. 1–43, 2018.

[60] A. Alves et al., *Web Services Business Process Execution Language Version 2.0*, OASIS, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[61] F. Jrad, J. Tao, A. Streit, R. Knapper, and C. Flath, "A utility/based approach for customised cloud service selection," *International Journal of Computational Science and Engineering*, vol. 10, 2015.

[62] M. Anisetti, C. A. Ardagna, E. Damiani, and J. Maggesi, "Security certification-aware service discovery and selection," in *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012, pp. 1–8.

[63] M. Anisetti, C. A. Ardagna, F. Berto, and E. Damiani, "A security certification scheme for information-centric networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2397–2408, 2022.

[64] M. Anisetti, F. Berto, and M. Banzi, "Orchestration of data-intensive pipeline in 5g-enabled edge continuum," in *IEEE World Congress on Services (SERVICES)*. IEEE, 2022, pp. 2–10.

[65] M. Anisetti, F. Berto, and R. Bondaruc, "Qos-aware deployment of service compositions in 5g-empowered edge-cloud continuum," in *IEEE CLOUD*. IEEE, 2023 (to appear).

[66] J. H. Yahaya, A. Deraman, F. Baharom, and A. R. Hamdan, "Software certification from process and product perspectives," *IJCSNS*, vol. 9, no. 3, pp. 222–231, 2009.

[67] F. Baharom, J. Yahaya, A. Deraman, and A. R. Hamdan, "Spqf: software process quality factor," in *Proc. of ICEEI 2011*, Bandung, Indonesia, July 2011.

[68] S. M. Darwish, "Software test quality rating: A paradigm shift in swarm computing for software certification," *Knowledge-Based Systems*, vol. 110, pp. 167–175, 2016.

[69] M. Anisetti, C. A. Ardagna, F. Gaudenzi, and E. Damiani, "A continuous certification methodology for devops," in *Proceedings of the 11th International Conference on Management of Digital EcoSystems*, 2019, pp. 205–212.

[70] F. Xia, L. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101–1102, September 2012.

[71] C. A. Ardagna and N. Bena, "Non-functional certification of modern distributed systems: A research manifesto," in *Proc. of the IEEE Conference on Software Services Engineering*, Chicago, IL, USA, July 2023.

[72] M. Anisetti, C. A. Ardagna, and N. Bena, "Software test quality rating: A paradigm shift in swarm computing for software certification," *IEEE Transactions on Services Computing*, 2022.

[73] E. Damiani and C. Ardagna, "Certified machine-learning models," in *Proc. of the 46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2020)*, Limassol, Cyprus, January 2020.

[74] F. Yoshioka and I. N., "How do engineers perceive difficulties in engineering of machine-learning systems? - Questionnaire survey," in *Joint Intl. Workshop on Conducting Empirical Studies in Industry and Intl. Workshop on Software Engineering Research and Industrial Practice*, 2019.

[75] C. Ardagna, R. Asal, E. Damiani, and Q. Vu, "From Security to Assurance in the Cloud: A Survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, August 2015.

[76] M. Anisetti, C. A. Ardagna, E. Damiani, and P. G. Panero, "A methodology for non-functional property evaluation of machine learning models," in *Proc. of the 12th International Conference on Management of Digital EcoSystems*, ser. MEDES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 38–45. [Online]. Available: https://doi.org/10.1145/3415958.3433101

[77] J. Winkler and A. Vogelsang, "What does my classifier learn? A visual approach to understanding natural language text classifiers," in *Intl. Conference on Natural Language & Information Systems*, 2017.

[78] R. Gall, "Machine Learning Explainability vs Interpretability: Two concepts that could help restore trust in AI," 2018. [Online]. Available: https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html

[79] L. Hendricks, K. Burns, K. Saenko, T. Darrell, and A. Rohrbach, "Women also Snowboard: Overcoming Bias in Captioning Models," in *Computer Vision – ECCV*. Springer, 2018.

[80] Global Legal Research Directorate, "Regulation of Artificial Intelligence in Selected Jurisdictions," The Law Library of Congress, Tech. Rep. January, 2019. [Online]. Available: https://www.loc.gov/law/help/artificial-intelligence/index.php

[81] M. Schmitz, "Why your Models need Maintenance," 2017. [Online]. Available: https://towardsdatascience.com/why-your-models-need-maintenance-faff545b38a2

[82] C. E. Perez, "Modularity," pp. 1–15, 2018. [Online]. Available: https://www.deeplearningpatterns.com/doku.php?id=modularity

[83] D. Mairiza, D. Zowghi, and N. Nurmuliani, "An investigation into the notion of non-functional requirements," *Proceedings of the ACM Symposium on Applied Computing*, pp. 311–317, 2010.

[84] ISO/IEC and IEEE, "ISO/IEC/IEEE 24765:2010 - Systems and software engineering – Vocabulary," *Iso/Iec Ieee*, vol. 2010, p. 410, 2010. [Online]. Available: http://www.iso.org/iso/catalogue{_}detail.htm?csnumber=50518