CHOReOS Governance V&V policies and rules (D4.1)

Antonia Bertolino, Guglielmo de Angelis, Cesare Bartolini, Amira Ben Hamida, Felipe Besson, Antonello Calabrò, Flavio Corradini, Francesco de Angelis, Mario Fusani, Fabio Kon, et al.

ICT IP Project

Deliverable D4.1

Governance V&V policies and rules

http://www.choreos.eu

LaTeX template v. 1.14

Project Number	:	FP7-257178
Project Title	:	CHOReOS Large Scale Choreographies for the Future Internet

Deliverable Number	:	D4.1
Title of Deliverable	:	Governance V&V policies and rules
Nature of Deliverable	:	Report
Dissemination level	:	Public
Licence	:	Creative Commons Attribution 3.0 License
Version	:	3.1
Contractual Delivery Date	:	M12 – 30 September 2011
Actual Delivery Date	:	21 October 2011
Contributing WP	:	WP4
Editor(s)	:	Antonia Bertolino, Guglielmo De Angelis, Andrea Polini
Author(s)	:	Cesare Bartolini (CNR), Amira Ben Hamida (Petals Link), Antonia Bertolino (CNR), Felipe Besson (USP), Antonello Calabrò (CNR), Flavio Corradini (UniCam), Francesco De Angelis (UniCam), Guglielmo De Angelis (CNR), Mario Fusani (CNR), Fabio Kon (USP), Pedro Leal (USP), Francesca Lonetti (CNR), Daniela Mulas (CNR), Andrea Polini (UniCam), Sarah Zribi (Petals Link)
Reviewer(s)	:	Animesh Pathak (Inria), Valerie Issarny (Inria)

Abstract

This document presents an initial view of the framework under development for CHOReOS governance and V&V. The focus is on specifying policies and rules on which the framework will rely. After discussing the ULS-FI challenges which are more strictly related to WP 4 goals, we overview the preliminary architecture which will support governance and V&V. We classify policies supporting governance, and start discussing more relevant ones, concerning choreography roles and life-cycle. We also propose approaches for modeling and handling SLA-related requirements. We devote special attention to V&V-related governance aspects, and identify some policies to govern online testing, service ranking and scalability. The framework is still preliminary, in that governance is a transversal concern, and it certainly needs to be harmonized with the components and processes undergoing parallel investigation in the other CHOReOS WPs.

Keyword List

Choreography, Future Internet, Governance, Life-Cycle, Monitoring, Policy, Ranking, Registry, Role, Service Level Agreement, Testing, Ultra Large Scale, Verification and Validation.

Document History

Version	Changes	Author(s)
1.0	ToC	Antonia Bertolino, Amira Ben Hamida, Antonello Calabrò, Flavio Corradini, Francesco De Angelis, Guglielmo De Angelis, Fabio Kon, Andrea Polini, Sarah Zribi
1.1	Draft contents release of most of the chapters.	Antonia Bertolino, Amira Ben Hamida, Flavio Corradini, Guglielmo De Angelis, Mario Fusani, Francesca Lonetti, Andrea Polini, Sarah Zribi
1.2	Draft update	Antonia Bertolino, Amira Ben Hamida, Flavio Corradini, Guglielmo De Angelis, Andrea Polini, Sarah Zribi
1.3	Draft update	Antonia Bertolino, Amira Ben Hamida, Antonello Calabrò, Flavio Corradini, Guglielmo De Angelis, Francesca Lonetti, Andrea Polini, Sarah Zribi
1.4	Pre-release for internal review	Antonia Bertolino, Amira Ben Hamida, Cesare Bartolini, Antonello Calabrò, Flavio Corradini, Guglielmo De Angelis, Mario Fusani, Daniela Mulas, Andrea Polini, Sarah Zribi
2.0	Q.A. – Release for internal review	Antonia Bertolino, Amira Ben Hamida, Cesare Bartolini, Antonello Calabrò, Guglielmo De Angelis, Andrea Polini, Sarah Zribi
2.1	Added contribution about TDD, and minor restructuring of the ToC	Felipe Besson, Guglielmo De Angelis, Fabio Kon, Pedro Leal
3.0	PTC Release	Antonia Bertolino, Amira Ben Hamida, Cesare Bartolini, Antonello Calabrò, Guglielmo De Angelis
3.1	Final release	Antonia Bertolino, Guglielmo De Angelis

Document Reviews

Review	Date	Ver.	Reviewers	Comments
Outline	16 May 2011	1.0	n.a.	n.a.
Draft	31 August 2011	1.4	n.a.	n.a.
QA	5 September 2011	2.0	Animesh Pathak (Inria)	The internal review was received by Antonia Bertolino that uploaded it on the internal CHOReOS wiki.
PTC	30 September 2011	3.0	Valerie Issarny (Inria)	Annotations throughout, uploaded in the wiki

Glossary, acronyms & abbreviations

Item	Description
bSLA	Business Service Level Agreement
BPMN	Business Process Model and Notation
CEP	Complex Event Processor
CSM	Core Scenario Model
DoW	Description of Work
FI	Future Internet
GSLA	Global Service Level Agreement
IDRE	Integrated Development and Run-time Environment
IT	Information Technology
MDA	Model Driven Architecture
MOF	Meta-Object Facility
NF	Non-Functional
PMM	Property Meta Model
QoS	Quality of Service
S&T	Scientific and Technical
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TDD	Test-Driven Development
ULS	Ultra Large Scale
USDL	Unified Service Description Language
V&V	Verification and Validation
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WP	Work Package
WPL	Work Package Leader
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
WSPL	Web Services Policy Language
XACML	eXtensible Access Control Markup Language

Table Of Contents

List Of Tables

List Of Figures

1 Introduction

As stated in the project's Description of Work (DoW), CHOReOS aims at addressing the challenges posed by the Ultra Large Scale Future Internet (ULS-FI) by devising a dynamic development process, and associated methods, tools and middleware, to assist the engineering of software service compositions.

The overall S&T strategy is centered on the four work packages WP 1–WP 4, each focused on a key aspect of the CHOReOS dynamic development process, namely:

WP 1: on architectural style;

WP 2: on development methods and tools;

WP 3: on middleware support;

WP 4: on Governance and V&V.

In particular, this document is the first deliverable of WP 4, which investigates development and management strategies, defines policies and develops components necessary to establish and exercise governance in the CHOReOS wide heterogeneous inter-organization setting. Emphasis is on policies and rules related to service and choreography V&V, which needs a deep re-thinking of existing approaches. The more V&V moves from the laboratory towards on-line application, the stricter the grade of discipline and the agreed contractual procedures which need to be established among all the involved parties become.

In Deliverable D1.1 [35], we already discussed the importance of governance in modern SOA applications, and reported extensively on state-of-art solutions and tools. The content of Chapter 5 of [35] thus constitutes the baseline for WP 4, and is considered a necessary background to the present document. In that chapter we introduced several concerns requiring special attention from research, to which the activity of WP 4 will be devoted. On the one side, considering governance, we overviewed many existing solutions. However, we also noticed that a standard commonly agreed implementation protocol for governance tools is missing; moreover, policies, which are a corner stone of governance tools, need to be explicitly and formally defined. On the other side, we explained that one concern of SOA governance which has not yet received adequate attention is V&V (as also presented in [23]). Especially in the ULS-FI context we need policies and tools for supporting continuous on-line dynamic testing and monitoring.

Consequently, in this document, we start addressing the above summarised concerns. In particular, the activity of WP 4 is structured into 3 tasks, and this deliverable is produced within Task 4.1 on Run-Time Governance Enforcement, which kicked-off nine months ago. As stated in the DoW, the content of D4.1, titled "Governance V&V Policies and Rules", should cover:

- Definition of governance frameworks for V&V of ULS choreographies;
- Definition of rules and policies enabling V&V strategies in ULS choreographies.

Thus we present in the following our initial proposal of the CHOReOS governance architecture, and outline a large framework of policies and rules.

1.1. Reading key

As we stated in the DOW, in this WP we intend to deliver a set of policies, to be used as a reference for regulating the development processes, choreography and services run-time behaviour and performances. Clearly, such policies and rules, to be effective, need to be agreed upon and abided by all choreography participants. Nevertheless, during our research work on both the CHOReOS IDRE and on the governance and V&V framework, we understood that such policies should also take into account existing constraints and established practices and ways to interact. In other terms, an authoritatively imposed, but ungrounded, set of policies and rules would undergo the risk of remaining just paper-work, without producing the desired integration.

Therefore, the initial set of policies we outline in this document is meant as a live reference regulation, to be iterated upon on a continuous basis in cooperation with the whole CHOReOS consortium. In particular, we will need to take into account the results and developments of the other WPs to coalesce their views into the WP 4 policy framework. The revised set of policies and the supporting mechanisms will be then incorporated into the next WP 4 deliverables.

1.2. Roadmap

The document is structured as follows: in the next chapter we speculate on the ULS-FI challenges which are more strictly related to WP 4 goals.

In Chapter 3 we introduce the preliminary architecture which will implement the CHOReOS governance and V&V framework. Governance management relies on special registries, whose functioning refers to an extensive family of policies (Section 3.1). We also present the detail of foreseen components enabling V&V governance (Section 3.2).

In Chapter 4 we propose a policy classification framework (Section 4.1), and in Section 4.2 survey existing standard notations for Service Level Agreements (SLAs).

Chapter 5 focuses on policies supporting governance. We discuss first about role related (Section 5.1) and life-cycle related (Section 5.2) policies. Then, we propose in Section 5.3 approaches and policies for handling (analysis and modelling) of non functional requirements. The chapter is concluded (Section 5.4) with a summary overview of how the governance framework addresses the ULS FI challenges.

Chapter 6 attempts a first definition of specific policies and rules for V&V governance, which include Activation Policies (Section 6.1), Rating Policies (Section 6.2), Ranking Policies and Rules (Section 6.3), the latter also illustrated by an example (Subsection 6.3.4), Choreography Enactment Policies (Section 6.4), Test Cases Selection Policies (Section 6.5), and Ultra-Large Dimension Mitigation Policies (Section 6.6).

Finally, in Chapter 7 we draw conclusions and hint at future work directions.

2 Governance and V&V in Future Internet Environments

The Future Internet (FI) will certainly require a change to our way of conceiving, implementing and using software, although by reading recent reports and research roadmaps [48, 32] it is evident that analysts and experts still do not completely agree on, or are able to precisely predict, the shape of the FI. Nevertheless from the above mentioned documents some commonalities emerge. The importance of social aspects over merely technical ones is widely recognized. In particular four pervasive forces are considered to be relevant and impacting on the FI:

- stakeholder conflicts,
- changing infrastructure and socio economic context,
- governance and regulation,
- user focus/inclusion.

In the FI we need to conceive methodologies and approaches that will permit the smooth integration of independently developed pieces of software in order to derive and provide users with more complex services, according to their changing requests. Within the FI context any software will be by nature characterized as an Ultra-Large-Scale (ULS) software system, where a ULS system "*is ultra-large in size on any imaginable dimension*" [48], like in the resulting number of lines of codes, in the number of people employing the system for different purposes, in the amount of data stored, accessed, manipulated, and refined, in the number of hardware elements (i.e. heterogeneity), etc.

ULS systems can be further understood by considering the following characteristics:

- *decentralization*, both in terms of their composing elements and with reference to their development,
- *inherently conflicting, unknowable, and diverse requirements*,
- *continuous evolution and deployment*, as ULS systems will integrate new capabilities while operating,
- *heterogeneity, inconsistency, and unstable elements* as ULS system emerge from the integration of elements owned and controlled by different stakeholder,
- *erosion of boundary between people and system*, as people become a central element of the system itself, providing contents and suggesting evolution possibilities,
- *normal failures*, as software and hardware failures will become the norm,
- *new paradigms for acquisition and policy*, as well as for controlling and monitoring them.

The response that the CHOReOS project provides to the above complex demands posed by the FI is in part based on the introduction of *Choreography* specifications, so as to provide FI application developers with a higher level of abstraction and greater flexibility with respect to rather using single services. The project will develop a conceptual model and an infrastructure supporting the introduction of choreographies as a "tool" to mitigate the issues posed by FI and its ULS dimensions. Within the whole picture, WP 4 specifically aims at managing the challenges to governance and V&V activities and to relate them to the choreographic centric vision embraced by the project. In the following we shortly discuss such challenges, for governance first and for V&V next.

2.1. Challenges to Governance in ULS FI

Governance is the act of governing or administrating. It refers to all measures, rules, decision-making, information and enforcement that ensure the proper functioning and control.

2.1.1. CHOReOS Definition for Governance

In the context of Service-Oriented Architecture (SOA), there are several ways to define governance. We survey, in the following, definitions from the literature to define the concept of SOA Governance:

- Anne Thomas Manes Research Director at Burton Group defines SOA Governance as processes that an enterprise puts in place to ensure that things are done in accordance with best practices, architectural principles, government regulations, laws, and other determining factors [42]. SOA governance refers to the processes used to govern adoption and implementation of SOA, ensuring and validating that assets and artifacts within the architecture are acting as expected and maintaining a certain level of quality.
- According to Paolo Malinverno [41], SOA Governance is about having discipline and making sure that the very important decisions go through to appropriate people, and that these people have the appropriate input to make those decisions.
- SUN [55] states SOA Governance as the ability to organize, enforce, and reconfigure service interactions in an SOA.

Based on the above definitions, in CHOReOS, **governance can be defined as a set of processes, rules, policies, mechanisms of control, enforcement policies, and best practices put in place throughout the life-cycle of services and choreographies (from the design time to run-time stage), in order to ensure the successful achievement of the SOA implementation.** Specifically, SOA governance is realized through a cycle consisting of policy definition, auditing & monitoring, and finally evaluation & validation (see Figure 2.1). The functional and non-functional expected behavior (of services and choreographies) is expressed through the specification of policies. Policies define the rules according to which systems should behave. More specifically, **the concept of a policy has been introduced as representing some constraint or condition on describing, deploying, and using some service** [40]. When such constraints or conditions are agreed between two or more parties, they become a *contract*. Generally speaking, SOA policies can be distinguished between two main levels of governance: design-time governance (e.g., code conventions, metadata compliance), and run-time governance (e.g., SLA).

Thus, SOA governance in general, and *Choreographies Governance* in particular, ask for the definition of policies and supporting tools taking into account the characteristics of FI and ULS highlighted above. Particularly challenging is the definition of governance policies and mechanisms that need to be distributed and controlled according to a decentralized and neutral paradigm. Indeed, without a shared and neutral governance the inherent multi-organizational nature of the FI might probably result in a situation of chaos from which it will be difficult to organize the required complex business interactions.

Figure 2.1: Policy Life-Cycle

In order to ensure that the integration of independently developed pieces of software is successful, an effort in introducing standardized notations, interfaces, data coding and more recently also semantics (through ontologies) has been undertaken by SOA companies. Nevertheless companies have soon realized that standards alone are not sufficient to ensure interoperability. The socio-technical nature of the SOA world means that it is necessary to consider services as active entities that operate aside or replace humans. To achieve SOA interoperability, it is then necessary to put in place also some social organization to govern the interactions among the participating services, aiming at assuring that everyone abides by the agreed social rules.

In CHOReOS the governing rules and procedures should be established and enforced by *super-partes* bodies, which must be trusted and accepted by anyone. This is what SOA Governance is conceived for. Indeed, the apparent flexibility and ease of use of service-oriented applications can be only achieved through discipline and an enforced framework of rules, policies and processes.

So far, SOA Governance has been mainly pursued for achieving service integration within one organization. This is obviously too limitative for FI. In the future vision of services all around us that dynamically connect and disconnect, on demand, towards some business objective, SOA Governance must be meant as a comprehensive management umbrella under which effective interoperability across organizations and platforms is ensured.

However, the mere definition of decentralized policies does not solve per se the issues and challenges that the CHOReOS project will need to address; we then also need means for monitoring, assessing and enforcing policies and rules. This functionality will be made available through specific mechanisms included in the CHOReOS platform as detailed in the following chapters.

2.1.2. Governance Research Domains and Challenges

CHOReOS will investigate both *design-time* and *run-time* governance. Governance activities can be seen as a transversal layer that ensures the adoption of the right way of doing things, the right time

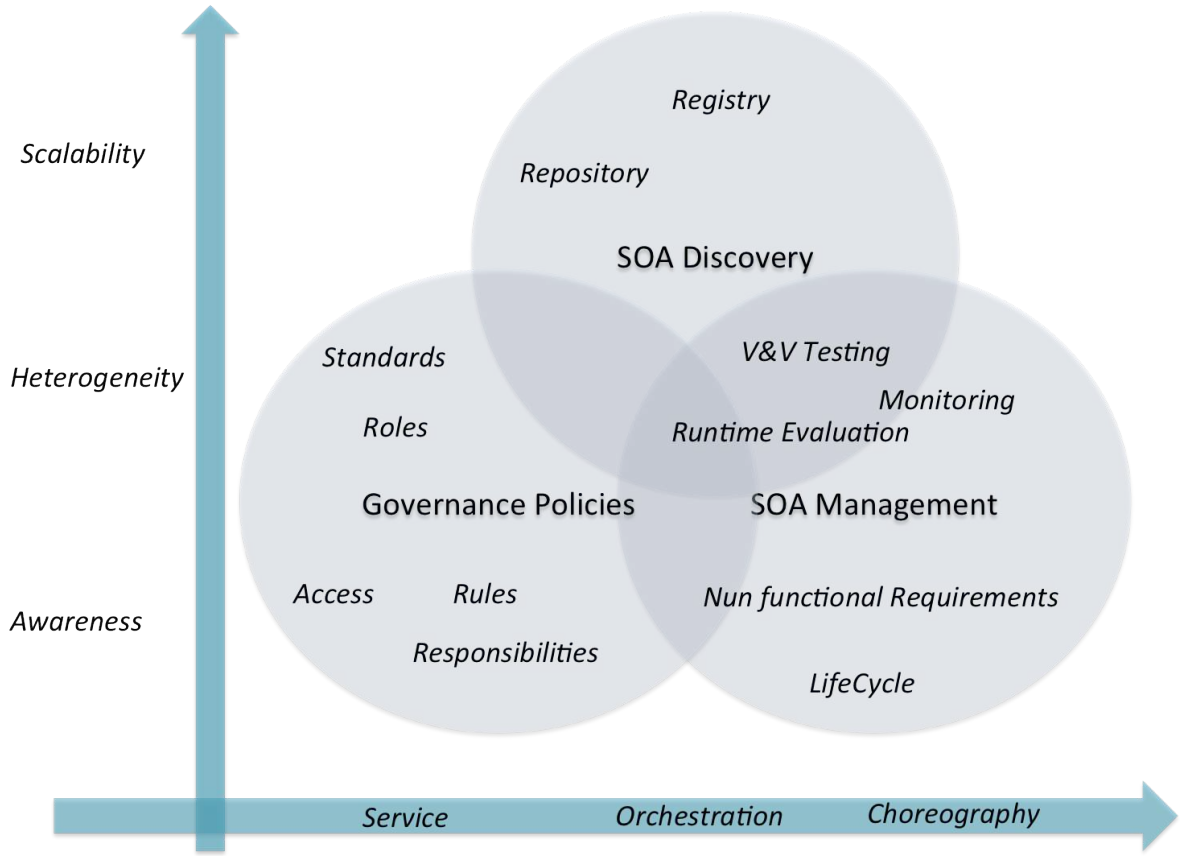

Figure 2.2: Overview of SOA Governance in Future Internet

and by the right persons. Governance is a paradigm underlying the whole service and choreography life-cycle and the IT system and at the borderline of three concerns: SOA Discovery, SOA Management, and Governance Policies (as illustrated in Figure 2.2).

- **SOA Discovery**: services registries and repositories provide service discovery capabilities at both design and run-time. Governance capabilities, such as looking for a service, retrieving it, or managing its life-cycle, are provided on top of a registry/repository mechanism. The service registry concern is addressed in the WP 2 of the CHOReOS project and is further extended in WP 4 by integrating governance capabilities for ULS choreographies. We expand on this in Chapter 3.
- **SOA Management**: covers the management of the service and choreography life-cycle from development until run-time. This includes the definition and management of the service and choreography at several stages of their life-cycle. Particular interest will be devoted to the V&V testing stages presented in Chapter 6. Moreover, SOA Management can also include the monitoring and run-time evaluation of the non functional requirements of services and choreographies. Consequently, it ensures the alignment between service consumer requirements and the run-time behavior of services. SOA Management needs to also address the choreography aspects by defining which rules, policies and standards are more relevant to be applied at different stages of the choreography life-cycle. The aspects of SOA management and governance are presented in Chapter 5.
- **Governance Policies**: policies are the cornerstone of the governance paradigm. Through the adoption of a common set of policies and standards, SOA governance makes the exposed services compliant with heterogeneous services coming from several platforms. These need to be

identified in order to implement the governance framework. In order to ease interoperability and service reuse, best practices and rules are adopted. Both SOA Discovery and SOA Management are concerned with governance policies as they define each step of the service life-cycle. Each stakeholder involved in the governance process has their roles and responsibilities that need to be identified. The governance process also resides in setting common service engineering conventions and standards. In the CHOReOS project we elucidate a list of governance policies and rules as being part of the governance framework. These need to cover the choreography concern and to face the ULS dimensions. In Chapter 5, we identify several rules and policies over different aspects such as the service discovery registry, the roles of the different stakeholders, the service and choreography life-cycle. In Chapter 6, we identify V&V related policies and rules.

2.1.3. How CHOReOS Deals With Governance in ULS FI

FI environments challenge the SOA Governance under several aspects such as scalability, awareness and high heterogeneity of services (presented as the vertical axis in Figure 2.2). The FI requirements can be accounted for at several levels of the realization of the CHOReOS Governance framework. We provide the governance registry enhanced with functionality for managing services and SLA life-cycles (presented in Chapter 3), V&V testing (presented in Chapter 6) and Test Driven Development abilities (the TDD will be presented in the next WP 4 deliverable). Moreover, we provide a list of policies and best practises for achieving governance and supporting Verification and Validation for Ultra Large Scale choreographies and services. In Section 5.4 we summarize the responses of the CHOReOS Governance Framework to FI challenges with regard to the several contributions.

2.2. Challenges to V&V in ULS FI

The clear definition of governance becomes even more relevant to make a new vision on V&V activities acceptable and practical, in a FI and ULS setting. Research in V&V approaches for FI constitutes another important aspect of WP 4, with particular emphasis on testing approaches. Many researchers and analysts have suggested that traditional software engineering activities should be at least partially (if not completely) moved to the on-line stage (i.e., during normal operation of a service). CHOReOS considers this indication as particularly compelling.

2.2.1. V&V Assumptions in ULS FI

With reference to testing we can note that the testing of software systems is traditionally structured as a three-stage activity, namely unit, integration and system testing. In each of the three stages, testing activities are based on some basic assumptions that in the development of “traditional” software are often left implicit, because commonly perceived as obvious. For our purposes we can list three main basic assumptions[1]:

1) *Software access*,
2) *Model/Specification availability*[2],
3) *Off-line experimentation*.

The first assumption foresees that in order to check the behaviour of the various modules, either in isolation or in agglomerates, the tester has the possibility of fully manipulating both the various elements composing the system, which he/she knows in advance, and its environment. Depending on the applied

[1] It is important to remark that such assumptions should not be considered as limitations, rather they simply denote a characterization of the testing activities within software development domains different from the service-oriented one.

[2] In the following with the term *models* we refer to either design artifacts, or any kind of specifications

testing strategy this assumption may go even further requiring the possibility of accessing the source code (in white-box testing).

The second assumption concerns the availability, before the system is put in place, of either some data, or behavioural models to be used for test planning. Considering specifically the different testing phases, this assumption concerns: the availability of models for single module during the unit testing phase, of models for module agglomerates during an integration testing phase, and of models for the whole system during system testing phase. In other words, the availability of a pre-run-time reference model is at the basis of many testing strategies both to guide test suite definition and to decide which is the correct result to expect for each test (in presence of a formal model possibly also to automatically derive the test cases). This assumption has been already questioned by "no-completely in-house" software development approaches. In particular in such contexts the unit testing phase ends up being penalized by the unavailability of any kind of models for the software provided by third-party. Nevertheless, often in such cases it is still possible to test the components using models directly defined by the integrators before run-time.

The third assumption refers to the fact that the software life-cycle generally foresees a pre-release stage in which both the system under development, and its composing elements, can be manipulated off-line within the selected testing environment. Any experiment carried on during this stage will not produce any permanent effect on the resources used by the system after the final deployment. Moreover, even after release, it is generally possible to continue to modify and evolve the system and experiment with it in a duplicated off-line environment without influencing the status and the behaviour of already deployed and running instances.

2.2.2. Challenges to V&V

In devising a testing approach within a FI setting, many of the assumptions foreseen by a traditional development process do not hold anymore. In particular, the three basic assumptions listed above are not anymore easily fulfilled and should be somehow relaxed, if not discarded.

Assumption 1 mainly affects those activities which are related to integration and system testing. A service developer could certainly test a service in isolation and within a controlled environment (in laboratory), but this is usually hard to apply for service agglomerates, since not all service interaction points are fully under the developer's control. Besides, given dynamic discovery and binding, it is also difficult to know in advance which external services a service under test will be bound to at run-time, and which will be the services that will participate at run-time to achieve the final common objective. Hence it is difficult to obtain trustable results by simulating their respective behaviours within a test environment.

Assumption 2 refers to the availability of some models useful for testing purposes. In a traditional setting the organization producing the application generally has a global view and control on what it is developing and integrating. So we can say that the final software application is to some extent the result of a coordinated and centralized effort. In the FI and SOA world this is no longer the case, on the contrary we can say that somehow removing such a central point of control is one of the objectives of the new paradigm. The service developer cannot know in how many different ways and within which compositions the service will be used. Due to the volatile composition mechanisms in place in a SOA setting (i.e. dynamic discovery and binding), the behaviour that a service should conform to, as well as its real usage context, can be fully identified only at run-time, while service integration is going to happen or is already in place. This fact suggests that, differently from "no-completely in-house" software development where testers can infer component models, in general strategies based on the derivation

of test suites at development time are not easily applicable. Thus, the adoption of strategies permitting to derive test cases on the base of on-the-fly techniques could result more effective.

Assumption 3 (the off-line assumption) is hardly applicable in the FI where services are made available to other parties just through the publishing of the service access points. Therefore even if we could assume to know all the services before run-time, without violating Assumption 1, in general the service to integrate is not available for off-line experimentation and the possible effects of a testing session will result in permanent effects on a running system unless specific countermeasures are taken.

2.2.3. How CHOReOS Deals With V&V in ULS FI

In consideration of the above,the CHOReOS project will investigate approaches and techniques for V&V on-line activities, which will permit to solve the highlighted challenges. In this deliverable, we mainly focus on the policies and infrastructure making possible the applicability of V&V activities at run-time, while in the next ones we will refine the infrastructure design and will release its components.

The very nature of FI and ULS systems ask the CHOReOS project to consider services not only as mere functionality. On the contrary the focus must go also to extra-functional properties. In particular the socio-technical nature of FI requires a special focus on trustworthiness. Indeed this is the direction that the CHOReOS project intends to follow and it is worth mentioning that on-line V&V activities are also targeted at deriving quality evaluation attributes for services, providers and choreographies as a whole, as detailed in Section 6.3.

3 Preliminary Architecture for Governance and V&V

In the previous chapter, we introduced the challenges brought by FI choreography-based applications. As we anticipated, to face such challenges, the CHOReOS project will implement a governance framework supporting policy-based choreography specification and its management. Special emphasis will be put on governance aspects related to V&V activities.

A first preliminary architecture of the CHOReOS governance framework is depicted in Figure 3.1. It includes several subsystems meant to ease services and choreography control and management from design time to run-time such as Rehearsal, the Test-Driven Development (TDD) [16] framework, which supports unit, integration, conformance, acceptance, and scalability testing for services and choreographies. The Rehearsal framework applies TDD to choreographies by automating multiple levels of testing such as compliance, integration, and scalability testing at development-time (i.e., offline testing). A description of the Rehearsal architecture, APIs, and its application on the CHOReOS development process have been reported in Deliverable D5.2 [17], and further details will be provided in the next D4.2.

Then, the Verification and Validation components handle the registration process and involvement of a service within a choreography. Finally, run-time quality evaluation ensures services and choreographies are behaving in compliance with the service level agreements previously contracted.

The central component of the framework is the Governance Registry, which is meant to provide a uniform way of governing business services. The classical functionalities of query and discovery implemented by service registry are here enhanced with the service level agreements management, V&V and TDD capabilities. The governance registry stands in fact as an interaction point for the V&V components and the TDD framework. Moreover, the governance framework interacts with the business service monitoring that enables the Run-Time Quality Evaluation presented in Section 5.3.3.

In order to adapt to distribution concerns raised by the FI environments and in order to achieve more flexibility and modularity in the CHOReOS IDRE, the governance framework is seen as a collection of several components. For each component, we provide an interface enabling governance capabilities at different phases of the service and choreography life-cycle. Details are provided in Deliverable D5.2 [17]. For the same reason, we separate the monitoring component for business service from the governance framework, but of course these are tightly interacting.

In the remainder of this chapter, we focus on the description of the high-level design of the V&V framework architecture and the CHOReOS governance-enabled registry as to provide a preliminary view of the main CHOReOS IDRE components that will be devoted to enable Governance and V&V activities. More detailed descriptions of the governance architecture components, as well as of their implementation, will be reported in the future deliverables of WP 4. In addition, Deliverable D5.2 [17] reports how the components of the CHOReOS Governance Framework relate to the others elements of the CHOReOS IDRE.

3.1. Governance Registry and Policies

An important component of SOA Governance is the Registry and Repository functionality, see e.g. [28]. Having a robust Registry/Repository promotes the discovery and reusability of services. Registries

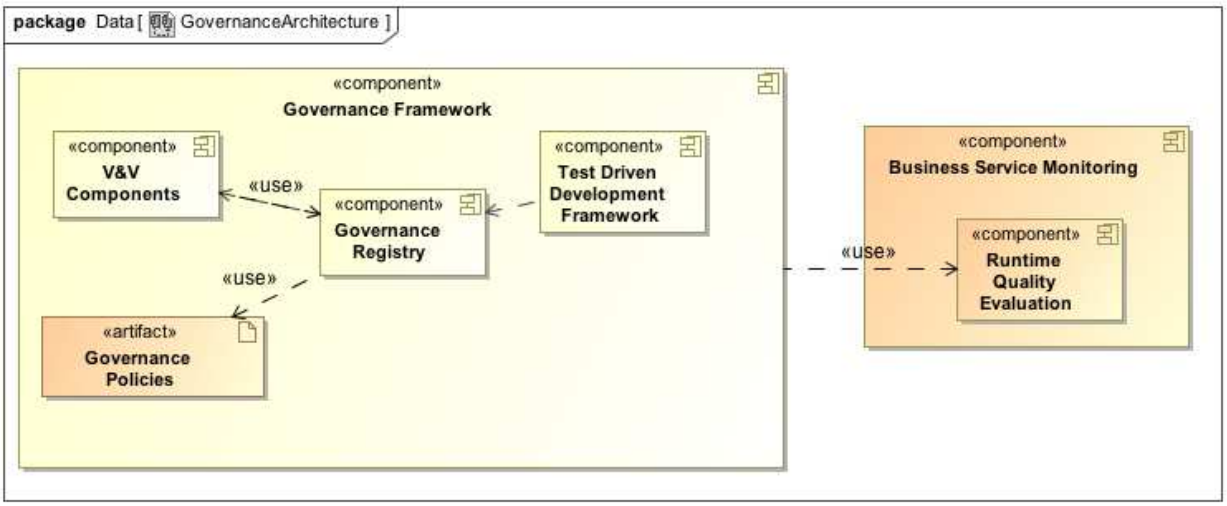

Figure 3.1: Governance Preliminary Architecture

serve not only to inventory and catalog service data, but also as places to store metadata about services, necessary to SOA Governance. These metadata go beyond WSDL documents and include descriptions of their functionalities, capabilities, and the locations of their service contracts. They may also include testing-related information (see, e.g., Section 6.5).

The CHOReOS project will provide registry/repository functionality both for services and choreographies. Service providers as well as choreography designer will publish services and choreography on such registries. Consumers looking for services and choreographies can refine their search according to non-functional concerns: for example performance, usage frequency, ratings, etc.

In order to make the CHOReOS Governance Framework adapted to FI challenges, design measures need to be taken into account. Indeed, we need to be able to manage an increasing number of services, users and policies. Besides, there is also the fact of dealing with a large quantity of heterogeneous policies and services.

In Figure 3.2 we present a preliminary architecture of the governance registry in CHOReOS. It is at the center of all the governance activities and it is also linked to the run-time middleware for services.

Governance Registry : In the CHOReOS context, the registry functionality is essential since a very large number of services coming from different sources need to be discovered and governed. Within WP 4, we provide a governance registry enabling the management of business services, which integrates the CHOReOS extensible service discovery registry (see [59]).

The governance registry allows the discovery of business services at both design and run-time. First, as we details in Chapter 5, it provides the ability of managing the business service life-cycle, the creation of service level agreement and their negotiation. It is also enhanced with Verification and Validation and Test driven development functionality. A status for the business service is assigned and evolves as the service is being developed, tested, verified and validated.

Second, the governance registry presents also a run-time view of the deployed services. Business services running in the service access middleware, and precisely in the distributed service bus nodes, are discovered and monitored. This way the governance framework is also able of evaluating at run-time if a negotiated service level agreement is respected or not. The governance registry reconciles both environments covering the service and choreography life-cycles from design to run-time.

Third, in order to tackle heterogeneity issues of business services coming from different sources, the governance registry relies on a uniform and common service description language. The

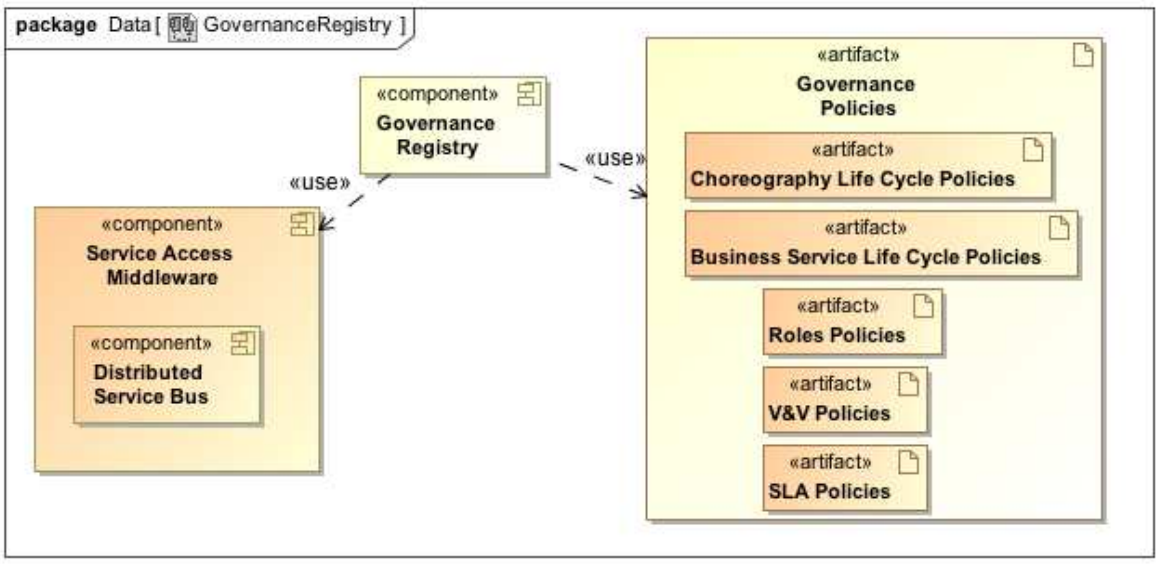

Figure 3.2: CHOReOS Governance Registry

USDL [3] enables the expression of the common business services descriptions in a unique agreed way.

The governance registry is dedicated mainly to business services but can be enhanced in order to provide a repository of choreography templates. Partners may be interested in being involved in one of the discovered choreographies. Choreography and V&V Policies are then applied. In the following we briefly present the governance policies that are supported in the CHOReOS governance framework.

Governance Policies are composed of the following policies:

Business Service Life-Cycle Policies are responsible for controlling the life-cycle of a service from its design to its deployment on the middleware. They ensure and guide the adoption of the best practises for governing the development process of a service. These functionality are addressed in Section 5.2.1.

Choreography Life-Cycle Policies are responsible for setting the good principles and the best practices for the different stages of a choreography life-cycle. This functionality is linked to the governance registry as choreography can be published and deployed on a run-time environment. This topic is addressed in Section 5.2.2.

SLA Policies refers to the policies that concern the service level agreements. These are related to their definition, publication, negotiation and monitoring at run-time. The CHOReOS governance registry enables the handling of the SLA life-cycle. These aspects are discussed in Section 5.2.3, Section 4.2, and Section 5.3.

Roles Policies define the responsibilities for each person or application with regard to the governance framework. It states exactly who can do what and when. This functionality is essential for governance activities. The different identified roles interacts with the governance registry and are able according to their access permissions to operate functions. This aspect is addressed in Section 5.1.

V&V Policies the Service Registry adopted within the Governance Framework will augment the discovery and directory service functionality with testing capabilities. Here, the idea is that a service can be tested at the time it asks for registration. More in general, such testing activities can be extended to the whole life-cycle of a service. In the literature such approaches are usually referred as "on-line testing" [22][20]. Approaches belonging to this class can be differentiated mainly on the basis of the information used to carry out the testing session. The main advantages in enhancing the functionality of the service registry with on-line testing approaches is that in those scenarios, integration tests are executed in the real execution environment providing more realistic results. In this way, only "high quality" certified services will be guaranteed to pass the registration. In Chapter 6, we expand on the policies regulating the V&V activities within the CHOReOS Governance Registry.

3.2. Components Enabling V&V Governance

The CHOReOS Governance Framework will constitute a sort of "control panel" for the management of ULS FI choreographies, providing components for proactive guidance and control of their composing services. It includes the enhanced registry discussed above. It will also support choreography-oriented testing approaches to be applied both at development time, and at run-time for the verification and the validation of services that declare to play a role within a service choreography.

With respect to the techniques applied at run-time, the governance framework will include both monitoring components, in which services behaviour and provided QoS are monitored during real service execution, and on-line testing strategies, in which testing executions are activated at run-time in order to expose possibly/suspected misbehaving services with respect to the specified choreography.

With reference to Figure 3.3, in the following we provide a brief description of the main components currently under development. Notice that these components include the traditional components needed in any test environment, such as the Test Driver, the Test Oracle, etc, plus other new components which are specifically conceived for enabling run-time testing of services interacting within a choreography, including the Reputation Center, the Choreography Participant Testing, the Run-Time Policy Monitor.

Governance Registry: we already introduced in Section 3.1 the CHOReOS Governance registry. As originally foreseen in [21], we enhance this registry with testing functionalities and mechanisms to manage the installed testing handlers. We conceive these handlers as mechanisms permitting to modify a service registration procedure with additional functionalities. In particular testing handlers activate testing sessions on services for which a registration request, or a modification of the associated entry, is received.

Test Suites Repository: this component permits to store and index test suites so that they can be executed to assess running services. Test suites will have to be defined following defined coding conventions and structural frameworks that will defined by the CHOReOS project.

Test Driver: this component, of which several different instances will be available and dispersed over the IDRE, permits to retrieve test suite from the Test Suite Repository and to execute them on a service. The driver is agnostic with respect to which test strategy is applied and which test cases will be launched; it is activated by the testing handler, which provides the necessary information to identify the service to test and the test suite to execute.

Test Oracle: this component permits to assess if the outcome of a test invocation made by the test driver is acceptable with respect to what was expected.

Policy Repository: this component permits to store policies governing the usage and execution of the various elements in the V&V infrastructure.

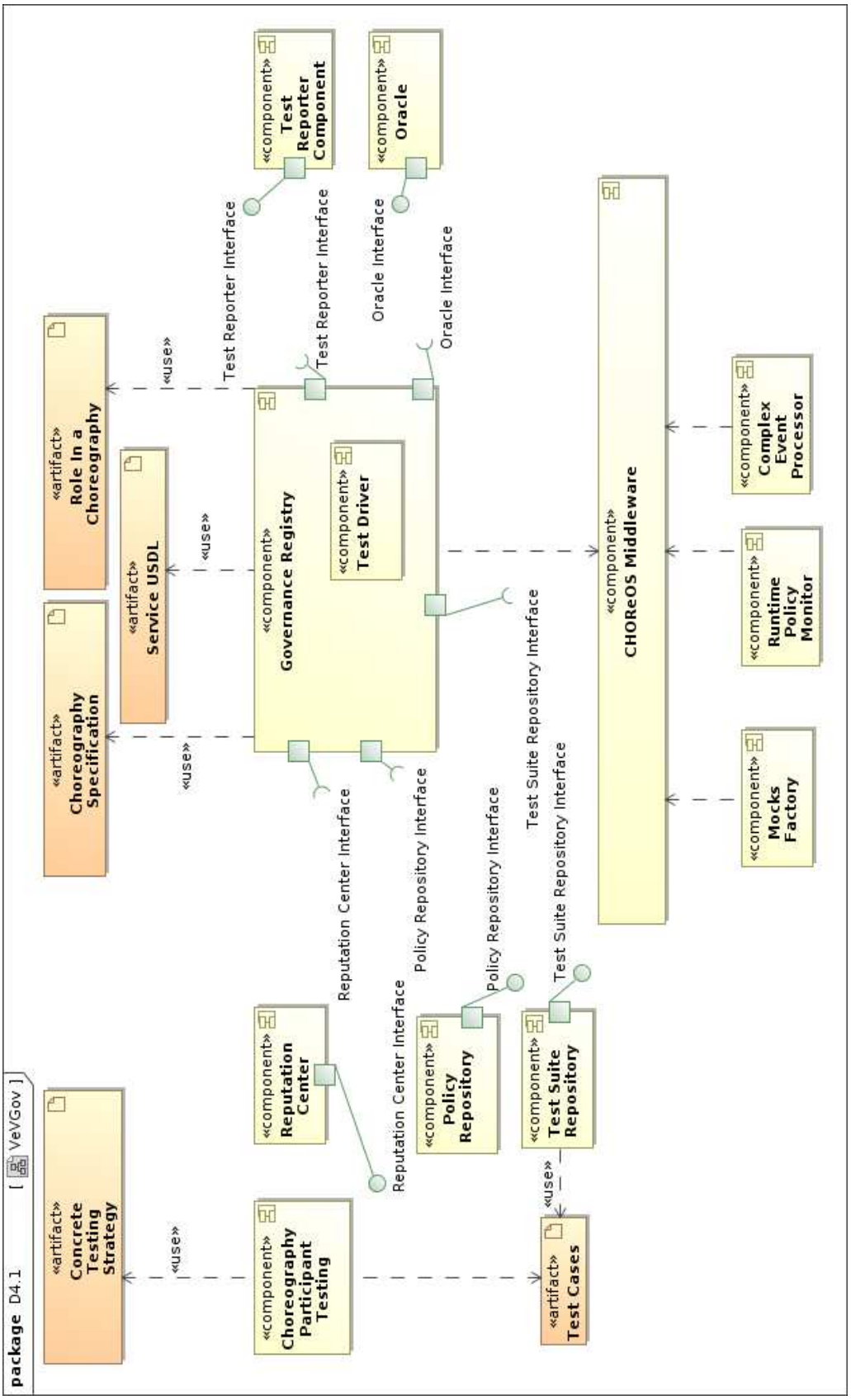

Figure 3.3: Preliminary Architecture of the V&V Framework

Test Reporter: this component permits to store the results of a launched testing sessions. Information reported can be used by governing authorities to put in place inclusion/exclusion policies for services (and their providers), based on the results. In principle only services successfully passing the testing sessions should be admitted.

Mocks Factory: this component permits to derive proxies and mocks necessary to test services willing to participate to a choreography. Created proxies and mocks permit to assess also how a service is able to interact with the roles specified in the choreography.

Reputation Center: this component logs information concerning reputation of services belonging to the CHOReOS Governance Framework. In trust management systems [38], reputation provides a measure of trustworthiness based on the referrals or ratings from members in a community. In CHOReOS V&V framework, reputation is a useful piece of information to improve the service selection process and to put in place policies concerning service life-cycle activities related to those services made available by the corresponding service provider. Hence, the Reputation Center is in charge of evaluating reputation metrics for enactable choreography. The evaluation should be based on the evaluations of single reputations for the possible participating services.

Choreography Participant Testing: this component permits to automatically derive test cases from choreography specification. Specifically, taken as input a choreography specification C, for each roles A that C defines, this component derives a test suite that can be executed in order to asses if a service can actually play A when integrated in C

Complex Event Processor: this component (CEP) analyzes the messages exchanged through the CHOReOS middleware (i.e. primitive events), and infers complex events. Specifically, it is a rule engine which monitors defined rules and policies in order to detect possible violations. When a match with a complex event is detected (which means a violation of a policy occurred), the event is notified into specific channels of the middleware.

Run-Time Policy Monitor: this component is the orchestrator of the overall monitoring architecture. It manages the CEP, and configures the communications between the monitoring architecture and the middleware. Specifically, the Run-Time Policy Monitor fetches the governance policies described in Section 3.1 in terms of monitoring requests, analyzes them, and configure one or more rules on the CEP. Then, it instructs the CEP in which dedicated channel of the middleware the CEP has to notify the verification of a complex events. Also, the it redirects the users of the monitoring architecture (i.e. consumers) to a proper the notification channel.

4 Policies for Service-oriented Systems

Policies are increasingly used for managing service-oriented systems. Indeed, policy-based management on one side supports dynamic adaptation, because by modifying the policies, the system behaviour will change as a consequence, and on the other side it allows for tuning control on large scale evolvable services compositions, because the control is embedded within the policy rules, and therefore is naturally distributed.

In this chapter we introduce policy concepts, classification and notations, before discussing governance and V&V policies in the subsequent chapters.

4.1. Policy Classification for CHOReOS Governance

Governance policies yield an eminent position in SOA management, and have been actively discussed in the related literature. Notably, the OASIS Reference Model for Service Oriented Architecture [40] deals extensively with policies and contracts for the management of services.

Although of course governance at the level of a single service remains relevant, within the CHOReOS project we are especially interested in governance at the level of the choreography, i.e., concerning those policies that regulate the interactions between the services composed within a choreography.

Policy frameworks have been widely studied in the early 90's with reference to policies for the management of complex distributed systems. More recently, they have been applied to service-oriented systems, e.g. [47, 62]. Thus certainly we do not want here to re-invent the field from scratch; on the contrary, we can look at the conceptual models and approaches which have been early defined and adapt them to the CHOReOS context.

Policies can be specified at various different levels of abstraction, and correspondingly will impact different aspects of system management. Such levels can be organized into a hierarchy [44], which somehow implies a policy transformation process from the higher -more abstract- policies towards the lower -more concrete- policies. The transformation corresponds to a stepwise refinement from human-targeted goals to concrete, executable procedures.

Concerning systems management, reference [44] identify six levels. We recall them below, going from abstract to concrete:

1) Societal policy (principles), in essence prescribing modes of conduct of humans;

2) Directional policies (goals), stating for example organizational or corporate goals;

3) Organizational policy (practices), which translate goals into plans and quality programs;

4) Functional policy (targets), refining practices into functions to be accomplished, such as integrity requirements, quality measures, and so on;

5) Process policy (guidelines), specifying the processes to be supported, for example automated quality tracking;

6) Procedural policy (rules), which consist of the derived executable procedures.

The above hierarchy is generic, and can be applied to the management of service-oriented systems, and of choreographies as well. So, for example, we may establish policies regulating proper models of behaviours for human users accessing service choreographies, and so on. **In the scope of CHOReOS, we will mostly focus on the two lower levels of the hierarchy (i.e., item 5 : Process policies, and item 6 : Procedural policies), considering those policies which dictate the modes and responsibilities in the interactions among services.** In particular, in this deliverable we introduce process policies (i.e., guidelines), whereas in the next deliverables we will work toward instantiating these into procedural policies, or rules.

Policies may vary widely not only concerning their abstraction level (i.e., vertically along the hierarchy), but also concerning the aspects and targets which they regulate (i.e., horizontally at one level of the hierarchy). Therefore, following [56], before starting to define CHOReOS policies and rules, we introduce a classification framework through which all aspects involved into definition and management of policies can be collected into one comprehensive classification, capable to cover all hierarchy levels.

Although the importance of establishing a governance framework for enabling collaborative V&V in multi-stakeholder service compositions is hinted at by some authors (e.g., [18, 61]), we were not able to find any existing framework structuring the necessary policies. Therefore, we have started establishing a preliminary framework from a survey of existing policy-based SOA governance frameworks and consideration of the CHOReOS conceptual model.

To help structuring these aspects and concerns, we will refer to the very intuitive 5 W's and 2 H's analysis framework[1]. With reference to V&V governance, such W5H2 framework would cover:

Who? Policies should define the stakeholders involved in performing and enforcing the V&V governance on one side, and in abiding by the established rules on the other.

What? What is the aspect that is regulated by the V&V governance? For example, this could refer to functionality, QoS or non-functional properties, or standard compliance [49].

When? When should V&V activities be carried out?

Where? Where is V&V performed, with reference to the scope of policies application and also the service choreography deployment platform.

Why? Policies can also establish the V&V objectives.

How? In order to enable V&V activities, policies could also be established to requested procedures and artifacts required for certain verification and testing strategies.

How many? Establishing how much testing, or the frequency of monitoring are crucial points of V&V management. For ULS choreographies this aspect becomes even more important.

Wies has previously proposed a classification scheme for policies [56], which at the time was not conceived considering services and choreographies. However, as we said before, the concepts remain the same and hence we have reasoned on how Wies classification could be applied and adapted to CHOReOS governance scope. The resulting CHOReOS classification scheme for V&V policies is depicted in Figure 4.1. The diagram depicts several axes structuring the conceptual domain of policies along different dimensions. On the axes we report labels for the various categories of each dimension that a policy could belong to (the axes only measure on a nominal scale). This categorization exercise thus results into an abstract radar diagram which provides CHOReOS first, preliminary policy classification. It is also interesting to attempt a mapping of the above discussed W52H framework, which is intuitive but quite abstract, onto the same classification scheme. We aggregate the axial dimensions into the W and H concepts and illustrate this in the figure by the circling the related areas within coloured ovals surrounding the axes tags (see the legend in Figure 4.1).

[1] The W5H2 framework analyses a fact or problem or concern by answering to the set of questions: Who, What, When, Where, Why, How, How many.

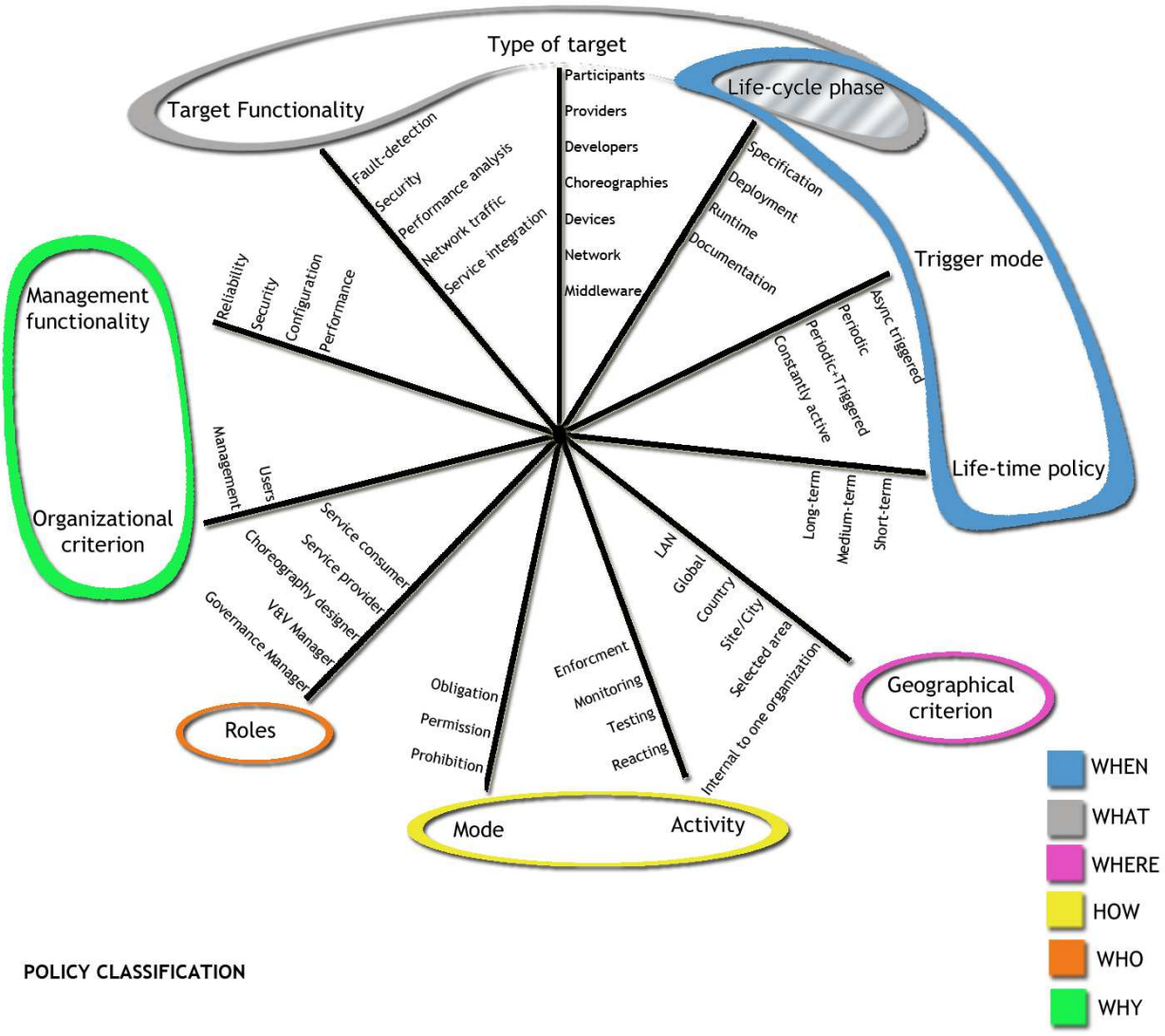

Figure 4.1: Criteria for V&V Policy Classification

In the remainder of this chapter and in the next one we start defining and discussing policies, which are relevant in the context of the CHOReOS project. As said, with reference to the six abstraction levels previously introduced, in this deliverable policies are generally provided at the “process” level.

4.2. Policies and SLA Standards for Governing Choreographies

In this section we first provide a short survey of languages commonly used for expressing policies and SLAs, and then summarize the standards that we adopt in CHOReOS.

4.2.1. Survey of Commonly Used Policy Languages

Several languages have been created for defining both policies and SLAs. In the following, we give an overview of the most significant ones.

eXtensible Access Control Markup Language (XACML)

XACML is probably the ancestor of policy languages. Created by the OASIS XACML TC in 2003, version 3.0 is being developed at the time of writing this document. XACML is an XML tagset, with implementations generally targeting most Java servers.

XACML rotates around the concept of *resources*. A resource is anything that can be accessed, such as data or a service. Resources can be accessed by *subjects* in various ways. Access types are defined as *actions*.

Resources, subjects and actions are the building blocks of a *target*, which is a construct that defines the subjects it applies to, the resources it binds, and the actions it enforces. A target represents a subset of the space of all possible combinations of resources, subjects and actions. To verify if a subject trying to gain access to a specific resource is an element in the target's subset, XACML uses boolean conditions.

Rules define the policies proper. Simply put, a rule can be applied only if the access attempt belongs to the space of its target (defined in XML as a child of the *rule* element). Rules are based on boolean *conditions*, and the possible outcomes of the application of a rule is an answer which is *permit* or *deny*. Any number of rules can make up a *policy*, which also has a target to determine if it must be applied. A policy can be the root of a specification, or it can be nested into a larger *policy set*.

Several nodes participate in an XACML interaction. The most relevant are the *Policy Decision Point* (PDP) and the *Policy Enforcement Point* (PEP). When an action is requested on a resouce which is protected by a PEP, this creates an XACML request, sending to the PDP information about the subject, action, and resource. The PDP will compare the request with the policies known to it, and if it finds some policy that applies to that target, it evaluates its rules and sends a reply to the PEP. The PEP then responds to the requestor by granting or denying access to the resource based on the PDP's reply.

So far, XACML does not seem fit to accommodate quality of service, only admission control. However, at least basic QoS requirements can be addressed using admission control. For example, the rules might be based on information on network traffic or server load; such information can be provided by the PEP when creating the XACML request.

Web Services Policy Language (WSPL)

Developed shortly after XACML itself, WSPL is a "XACML profile for web services" [11]. The developers of this standard describe it a a subset of the XACML tagset [12]. In short, WSPL uses the same syntax, structure and tags of XACML, but adopts additional constraints aimed specifically at specifying policies for web services.

In a nutshell, a policy set in WSPL can only target a web service. Additionally, each policy must describe a specific *aspect* of the policy set. For example, one policy might focus on the "service charges" aspect, another policy might address the aspect of "available bandwidth", and a third might control the "user priority" aspect. The combination of the policies of all the aspects of a web service defines its overall policy set.

The main purpose of WSPL is to combine policies. If a service provider and a consumer (which might be another services) want to interact, the provider will have a policy which describes the guarantees it *offers*, and the consumer will have a policy which describes the guarantees it *requires*. The two policies can be combined for negotiation, so that the partners can determine whether one's policy meets the other's requirements, and, if there is more than one policy which satisfies its needs, the merging can be used to select the preferred one.

Web Services Policy (WS-Policy)

A more recent standard than XACML, the Web Services Policy language [53], or WS-Policy (not to be confounded with WSPL, described in the previous section), was developed by the World Wide Web

Consortium (W3C) in 2006. It is composed of two separate XML specifications, WS-Policy and WS-PolicyAttachment (the latter is defined to allow compatibility between different versions of the WS-Policy and WSDL standards). The purpose of this standard is to enhance the interaction between a service provider and a service consumer by giving additional information (beyond the basic ones normally specified by the WSDL descriptor), on how the interaction should be carried out. To make it simple, this language aims at converting information which would normally fit in a documentation into a machine-readable format.

Such information takes the name of *metadata*. Metadata enhance the specification by adding details about how an invocation should be constructed, additional protocols it should use, and so on. However, metadata are not part of the WSDL specification and would be ignored by softwares not able to understand them. The purpose of metadata is the following: whereas a generic tool for creating SOAP invocations would fail against a service because it has specific constraints on how it must be invoked, a tool capable of interpreting the metadata provided by the service would be able to correctly generate SOAP invocations complying with those constraints.

The basic construct around WS-Policy is the *assertion*. A policy assertion is a bit of metadata which expresses a specific requirement for that service. For example, an assertion might denote the need for a specific security level, or the use of a certain protocol, and so on. Assertions are not defined in the WS-Policy standard, but rather left to developers of individual standards. The purpose of WS-Policy is not to define assertions, but to combine them and use them within a web service specification. The Web Services Policy Working Group provides a documentation on how to define policy assertions [58].

WS-Policy then combines assertions into expressions, using self-explanatory constructs such as *All* and *ExactlyOne*. A service consumer can access the service requiring that policy if the policy expression is satisfied. Additionally, policy assertions can be defined as optional, with the different purpose of allowing access to all customers but improving the service experience to policy-aware ones. In other words, access behaviours which do not meet the condition of the optional policy assertion are not denied, but will operate differently (for example, with a different degree of optimization). Also, assertions can be marked as ignorable, meaning that they will have no impact on the interaction with the consumer, but providing information about the service behaviour (for example, a service which logs all requests), thus allowing consumers to make a more conscious decision on whether or not use the service.

Expressions are included inside a *Policy* node, which in turn is attached to any part of the WSDL specification, thus protecting either the whole service, or just specific operations, bindings, or something else. The complete structure of the model of the WS-Policy standard is shown in Figure 4.2.

Similarly to what can be done in WSPL (as described previously), the policies of a service provider and a service consumer can be combined in WS-Policy, to determine whether the two requirements are compatible.

Web Services Agreement (WS-Agreement)

WS-Agreement [13] is a standard developed in 2004 by the Grid Forum committee. It differs from the standards described so far in that it does not define a policy over the quality that a service provider offers or a consumer requests, but establishes an agreement between the two partners. In other words, whereas WSPL and WS-Policy need only be defined on the service provider *or* the consumer to convey significant information, and the contractual agreement between the partners in obtained by merging two separate policies (as described with respect to WSPL and WS-Policy), WS-Agreement can not be defined on a single role, but requires a provider-consumer relationship.

In this sense, WS-Agreement aims at creating an implementation of the business concept of a *Service-Level Agreement* (SLA for short). By all means, WS-Agreement establishes a contract between the two parties, much in the same way that a legal contract is established between two subjects. The comparison goes well beyond a descriptive example, because the two partners in the establishment of an agreement are the *initiator* and the *responder*. Similarly to what happens in civil contracts,

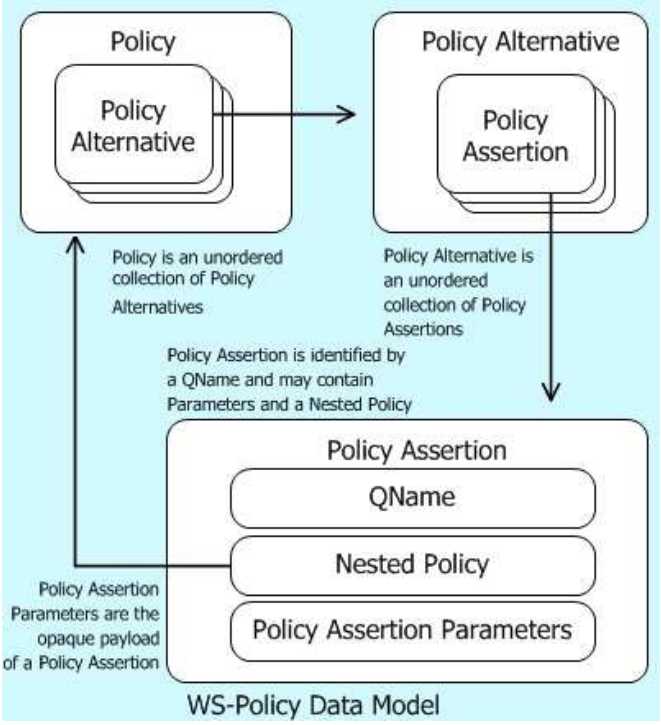

Figure 4.2: WS-Policy Data Model [8].

the initiator sends an *offer* to the responder, who evaluates the offer and replies with either *acceptance* or *rejection*.

Basically, the initiator could create an offer from scratch. However, to facilitate the process and avoid excess of rejections, the responder may, up front, provide a *template*. The template is a rough schema of the offers the responder is willing to accept, and contains a number of *agreement creation constraints*, which represent the rules the initiator must follow in creating his offer to avoid being rejected.

In this environment, there is no predefined association between the initiator and responder roles and the positions of service provider and service consumer. One of the purposes of WS-Agreement is to be symmetrical, meaning that the offer can come indifferently from the provider or the consumer. Similarly, WS-Agreement is designed to avoid being limited to some protocols or their versions, but rather it targets the very core concept of web services.

Figure 4.3 shows a sample structure of a web service interaction based on an agreement. The model is two-layered: the agreement layer contains a factory for creating the agreement based on some terms, and an interface to query the agreement about the current status; whereas the service layer contains the usual service application, with a factory to instantiate the service and a number of operations which can be invoked by the consumer.

The agreement itself takes the shape of an XML file, and it is made up of three sections:

- the name of the agreement;
- an optional context, which contains some metadata such as the names of the participants, the initiator and the responder, the duration of the agreement, and the template;
- the terms of the contract.

The terms are the core of the agreement. They can belong to the categories of *agreement terms*, *guarantee monitoring terms*, or *termination terms*. Agreement terms are the basis for establishing the agreement, so they are used during negotiation. Not all the requirements of one partner have the same degree of enforcement, and they can be either required or optional. At the end of negotiating a term,

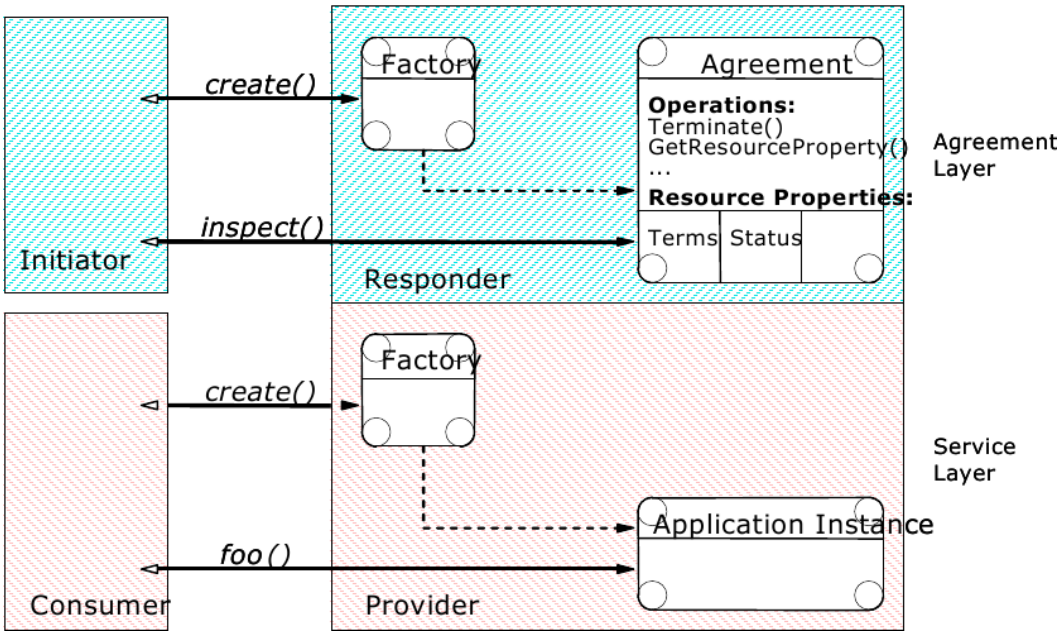

Figure 4.3: WS-Agreement Model [4].

this can be observed if it is met by the partners, ignored if it's not going to be used in the agreement, or rejected if a partner cannot meet the requirement. All this information is expressed through XML attributes.

Guarantee terms represent the quality level that must be maintained by the service provider. Guarantee terms are quite similar to the policies expressed by WSPL or WS-Policy. For example, they can represent a minimum CPU allocation, bandwidth, service response time and so on. This requirements are referred to as *service level objectives* (SLO).

Monitoring terms are used during the life-cycle of the relationship to determine whether the guarantees of the agreement are maintained. Basically, these terms express which values must be exposed to the initiator, and how this notification must occur.

Termination terms determine if and when a contract must be solved. They express the degree of defaulting which will not be tolerated by the other partner, therefore leading to the termination of the agreement.

Summing up, the life-cycle of a WS-Agreement is divided into three separate phases: a first phase represents the negotiation, where the agreement terms come into place, and the agreement is reached when the partners are satisfied over the compliance with the terms; then, the relationship is established with some degree of quality, and this degree represents the compliance with the guarantee terms; during the relationship, the monitoring terms are used to expose defaults in the service provider's quality of QoS, and if these failures to comply with the agreement matches the termination terms, the relationship is solved.

SOA-EERP Business Service Level Agreement (bSLA)

Following the idea of WS-Agreement, the OASIS organizazion in 2010 created a new standard, called the Business Service Level Agreement (bSLA for short), to support not policies of an individual role, but rather the policies underlying a web service relationship.

The structure of bSLA is more intuitive than that of WS-Agreement. In a nutshell, bSLA is an XML file which has a *BSLA* node at its root. The sections of the bSLA are *SLAParties*, *SLAParameters*, *SLAObligations* and *SLATerms*.

The section on the parties describes the subjects who participate in the relationship, namely the

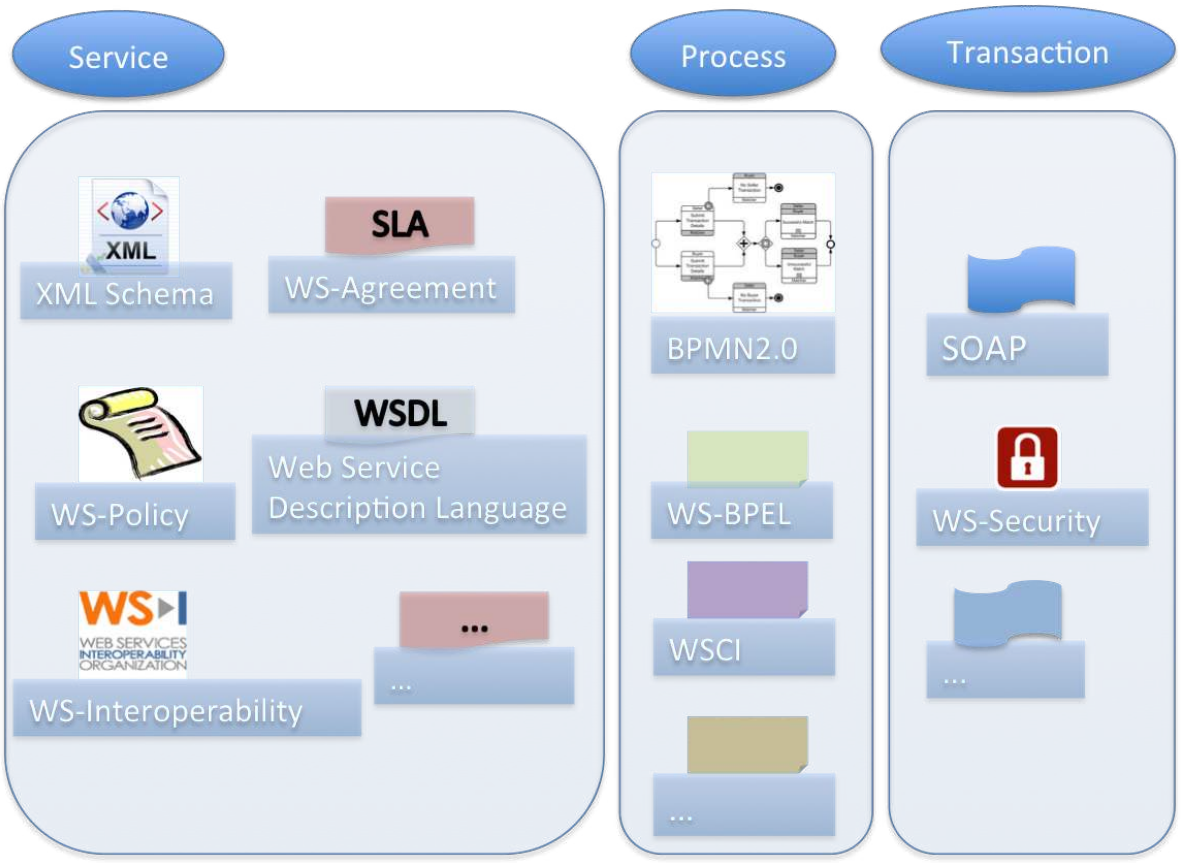

Figure 4.4: Service, Process and Transaction Standards

service provider and requester. There is no such concept as the initiator and responder in bSLA, since there is no separate negotiation phase (like the one used for the WS-Agreement).

The parameters section contains some basic metadata on the web service, such as the EndPoint Reference (EPR), costs, throughput and so on.

The obligations are the core of the bSLA specification. Basically, this section is made up of individual obligations, which collectively make up the SLO mentioned with respect to the WS-Agreement. Obligations are similar to the policies used in WSPL and WS-Policy, and may contain requirements on the availability of the service, a minimal throughput, and the like. Additionally, this section may contain some *action guarantees*, which describe what actions are to be taken in case the SLO is met or not met (generally, these are made up of fees in the former case or penalties in the latter). Since obligations and guarantees are limited in the bSLA specification, the agreement may contain an optional terms section which defines additional terms not expressed in the obligations section.

4.2.2. Suggested Policy Standards for CHOReOS Governance

The adoption of common agreed technological standards for software design and development eases software interoperability and helps addressing the scalability issue.

In the CHOReOS context, we deal with an important number of heterogenous services. Sources, protocols, development paradigms can differ from a service provider to another. The CHOReOS governance framework relies on a uniform service model based on USDL. Nevertheless, a good way to provide governance on top of such scalable systems is to adopt standards and rules. This eases the verification and validation of the used standards and their comparison with a common reference.

In Figure 4.4, we present a classification of some SOA policies and protocols according to the scope to which they can be related. We classify the level of policies adoption into three classes: Service, Pro-

Standard	Description	ref.
Unified Service Description Language (USDL)	USDL is a generic service description language consolidated from SAP Research projects. It aims to provide a way for users to model services from a business, operational and technical point of view. It defines nine modules related to each other to model of the overall service description: Service, Service Level, Legal, Technical, Functional, Interaction, Participants, Pricing and Foundation.	[3]
Web Services Business Process Execution Language (WS-BPEL)	BPEL defines a language for expressing web service collaborations. The BPEL is standardized and is commonly used.	[5]
Web Services Description Language (WSDL)	WSDL describes a web service and defines how it works. It defines a standardized syntax for expressing the most relevant information about web services.	[9]
Web Services Interoperability (WS-I)	WS-I defines a common way to expose business services allowing their interoperability. For example, the WS-I standard may set constraints on the WSDL definition.	[6]
Simple Object Access Protocol (SOAP)	SOAP is a protocol for XML-based messaging over a network as defined by its WSDL file.	[2]
Business Process Model and Notation (BPMN) 2.0	BPMN 2.0 is a standard that aims at providing a normalized graphical representation of business processes.([14])	[1]
eXtensible Access Control Markup Language (XACML)	this standard is based on XML and defines an access control mechanism on rules and conditions	[10]
WS-Agreement	this standard provides a normalized way of expressing service level agreements. An agreement is a contract between a service consumer (client) and a service provider essentially on QoS constraints such as latency or availability	[4]
Web Service Policy (WS-Policy)	WS-Policy defines a nXML model and syntax to express policies of a Web Service.	[7]

Table 4.1: Commonly Used Standards

cess and Transaction. Service policies are dedicated to the service level. Process policies concern the collaborations between several business services. It includes the service orchestration and choreography. Finally, the transaction level refers to the policies that can be evaluated at the level of the messages transmitted between services within a process. For instance, a service would abide to policies concerning its description as WSDL adoption. A choreography should respect the specification expressing it as for instance BPMN or WSC-I. Finally, the messages transmitted within a transaction should respect security concerns in order to avoid violation risks. Security concerns related to a message would be expressed in WS-security specification.

According to both the requirements of the IDRE elicited in Deliverable D5.1 [26] and the architectural style for service choreographies described in Deliverable D1.3 [36], the protocols and the artifacts referred within the Governance V&V framework will use most of the industry standard for Web Services. This is an explicit design goal that aims to develop a robust and wide interoperable framework, that can also enable Governance V&V activities on existing services with minor disruption.

In the following Table 4.1 we report the standards for services, SLAs, policies and rules that are commonly used by the SOA community and that we consider in our governance framework. Obviously other standards may be added as we develop the CHOReOS governance framework and as the choreography development process is implemented.

5 Governance Policies and Rules for ULS Choreographies in FI settings

The CHOReOS project depicts a highly distributed and dynamic environment where users and services from several platforms and infrastructures are able to interoperate. The CHOReOS Governance framework needs to rely on solutions adequate to FI challenges and choreography concerns. Therefore, it is compelling not to conceive a governance framework that imposes too strict obligations and restrictions. In fact, the application of strict restrictions and governance policies would not be easily applicable to the large number of services coming from heterogeneous sources and would lead to imposing too strict limitations to the free collaboration envisaged for the FI world.

Services involved in choreographies come from different heterogeneous and unknown sources. Hence, it would be difficult to govern them throughout their life-cycle. For instance, only run-time governance could be realized for a discovered service, whereas it would be possible to fulfill design time governance and V&V to a service being developed by known organizations. Consequently, the CHOReOS Governance Framework introduces as a set of governance tools rather than as a stand alone solution, used for enabling governance of choreographies and services at several steps of their life-cycle.

5.1. Responsibilities and Roles Policies

Notwithstanding the high level of automation and dynamism of service-oriented systems, people will always yield a central position and their operation remain determinant for assuring the successful performance of choreographies. Therefore, identifying the roles that people and groups of people assume within the CHOReOS governance framework is a necessary and crucial task [31]. People and groups make decisions in accordance to and within the constraints and rules stipulated by the governance policies. They can be divided into two groups: those who contribute to the establishing of policies and those who can be dictated by their application.

In the following we first identify choreography roles (i.e., the labels on the “Role” axis of Figure 4.1), and then outline the relevant use cases, including the tasks they should accomplish and their expected interactions.

5.1.1. Main Governance Roles

The CHOReOS Governance Framework distinguishes between the following roles: Governance Manager, V&V Manager, Choreography Designer, Service Provider, and Service Consumer. Specifically:

The **Governance Manager** manages the governance tool, identifies the roles and the rights for users, governs the stages for service life-cycle, and participates in defining policies. We illustrate Governance Manager’s tasks in Figure 5.1. The governance manager can operate both at the choreography and at the service level.

The **V&V Manager** is responsible for the definition and the implementation of the V&V strategy within the governance framework. For example the V&V Manager is in charge of: the selection of

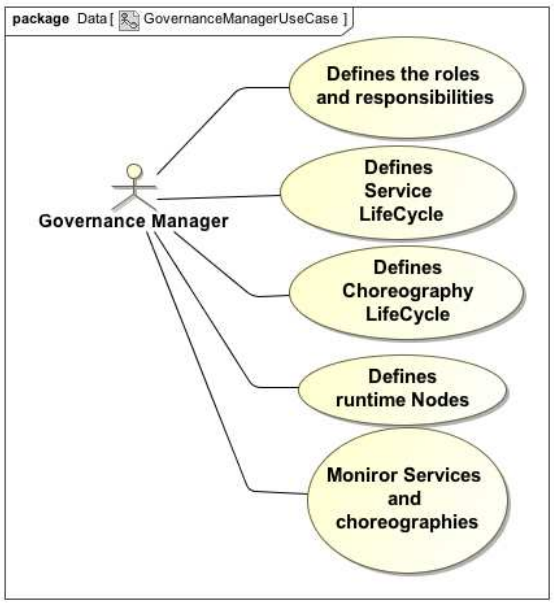

Figure 5.1: Use Case Modeling the Role of the Governance Manager

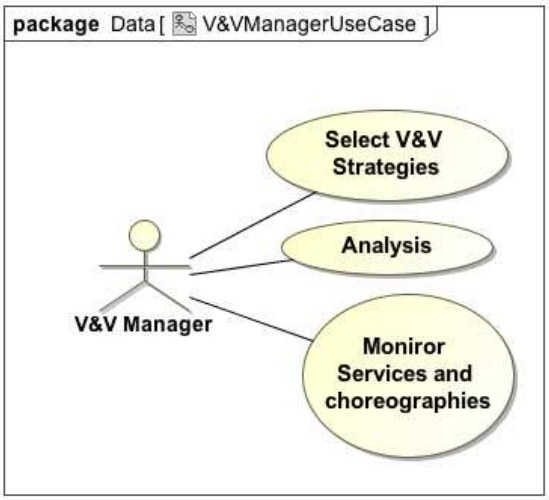

Figure 5.2: Use Case Modeling the Role of the V&V Manager

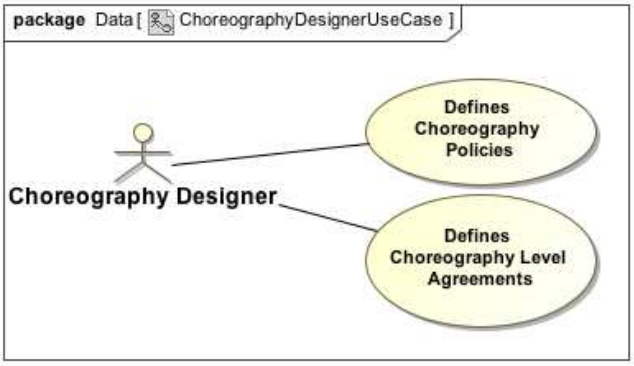

Figure 5.3: Use Case Modeling the Role of the Choreography Designer

suitable test strategy (e.g., driven by the choreography specification); the analysis and the management of the impacts of the V&V activities results on both the choreography and the service life-cycle; the definition the V&V policies that will be applied in the framework (e.g. how often V&V activities are performed or after which events). The V&V Manager as a general rule interacts strictly with the Governance Manager. Figure 5.2 depicts the V&V Manager functionalities.

The **Choreography Designer** is responsible of providing a choreography specification. For each choreography specification, the Choreography Designer may foresee a set of both policy template, and SLA template that actors playing the choreography have to abide by. The Choreography Designer may also define choreography level agreement with regard to QoS properties. Figure 5.3 depicts the Choreography Designer functionalities.

The **Service Provider** designs services, develops their implementations, describes their SLA contracts (which should be compatible with choreography level ones), performs tests, and contributes to the development of policies. The tasks are depicted in Figure 5.4. A Service Provider may be aware of a service choreography and decide to design and develop new services in order to fulfill the needs, and the business requirements specified in it.

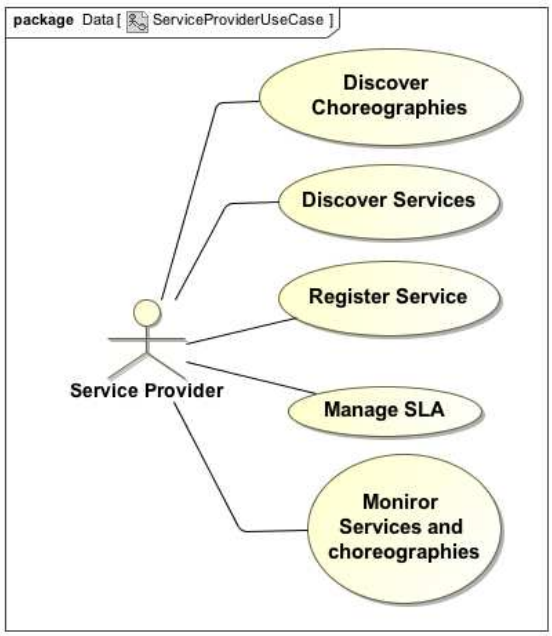

Figure 5.4: Use Case Modeling the Role of the Service Provider

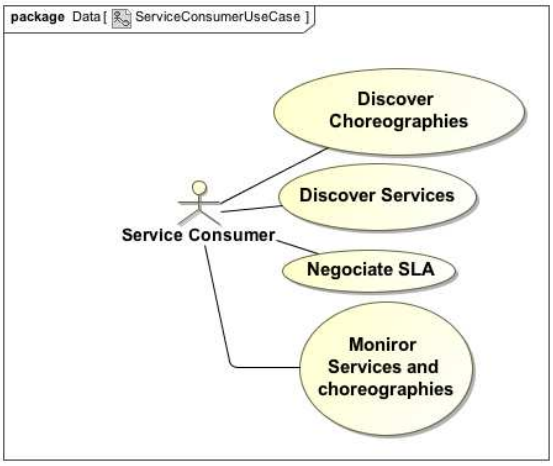

Figure 5.5: Use Case Modeling the Role of the Service Consumer

The **Service Consumer** (see Figure 5.5) discovers services in the repository/registry, uses the published services, and concludes a SLA contract with a Service Provider. The service consumer does not participate in establishing rules and policies like service administrator and provides, it is just dictated to use them. Figure 5.5 depicts the Service Consumer functionalities.

5.1.2. Main Governance Use Cases

As anticipated, we now introduce role-related process policies by means of some relevant use cases. In describing the following use cases, we identify the respective responsibilities of CHOReOS roles and the mutual interactions among them.

Choreography Registration Use Case When the design of a choreography is completed, the Choreography Designer can make it available by registering a specification of such choreography on a dedicated Governance Registry. From this registration, the Governance framework envisions that the lifecycle at run-time of the choreography is regulated by means of a set of policies specified by the Choreography Designer. In this context, such policies are agreed between the Governance Manager, and the V&V Manager. These policies define the rules under which a choreography is enactable, as well as the criteria and the parameters for evaluating the quality of a choreography.

The instantiation of such policies within the CHOReOS process are further discussed in Chapter 6.

Service Registration Use Case In CHOReOS architectural style [36], three different scenarios are described concerning the ways in which a choreography can be composed, which are “ad-hoc”, “role-based” and “requirement-based” (see [36], Chapter 5). The role-based scenario, in particular, foresees that Service Providers promote their services (i.e. either develop, or adapt) as participants “fitting” one or more roles to be played in a given choreography. This scenario is conceived to fit the case that a choreography has already been enacted, registered in the Governance Registry, and in the dynamic FI world, services can dynamically enter and exit the choreography, playing one of the specified roles. The same service can play different roles in different choreographies depending of the provided functionality. In other words, such scenario foresees that upon registration (or otherwise by updating a previous

registration) in the Governance Registry, the Service Provider specifies some role(s) the service will play and in which choreography(ies). From a testing perspective, this is the most challenging case, as the Choreography Designer needs to check whether the service can comply to the role, by testing for conformance to the specification.

In this sense, the registration of a service results as a critical point for the Governance Manager, and the Choreography Designer that wants to guarantee the registered services abide by both the functional and non-functional specification foreseen by a choreography. As an extension of the work proposed in [22], the Governance framework we are developing in CHOReOS supports V&V activities that aim at testing if a service implementation actually conforms to the role that the Service Provider claimed during its registration. Furthermore, in this scenario, it is equally important for the service registry to re-test (i.e. either periodically, or event-driven) a service that has been already registered in order to guarantee that its behaviour did not change over time.

Also for these use cases, Chapter 6 presents and discusses a set of policies that will be exploited within the CHOReOS project.

Service Monitoring Use Case A common practice in SOA, is that Service Consumer may negotiate the application of specific SLAs and specific policies, by interacting with the services offered by a given Service Provider. Furthermore, in a more extensive scenario, SLA as well as policies could be requested at the choreography level directly by Choreography Designer.

Thus, among others, an objective of the Governance Framework is to provide support for observing if such agreements or policies are actually honored. In other words, each actor identified in Section 5.1 should be able to define monitoring rules that focus on analyzing the events related to the life-cycle of a service, reporting any violation of the negotiated behavior that may occur. As described in Chapter 6, specific V&V policies could be applied reacting to a violation: for example deleting a service from the registry, or decreasing its rating.

Choreography Monitoring Use Case In addition to the scenario where the constraints are associated to a single service, a service choreography may agree on SLAs and policies that concern the interactions it specifies as a whole. Also in this use case, all the actors described in Section 5.1, but especially the V&V Manager, and the Choreography Designer, are interested in defining monitoring rules that observe how each enactment of a choreography is proceeding. The monitoring functionality of services and choreographies is provided by the Business Service Monitoring component.

5.2. Life-Cycle Management Policies

Defining the life-cycle policies of the resources to be governed is essential. Indeed, in the CHOReOS governance framework, we need to define the different steps of the life-cycle of business services, choreographies and Service Level Agreements. Governance activities apply policies and constraint rules at several steps of the life-cycle for ensuring the good behavior of these resources. The following sections are devoted to the description of the life-cycle management policies for services (Section 5.2.1), choreographies (Section 5.2.2) and Service Level Agreements (Section 5.2.3).

5.2.1. Service Life-Cycle Management Policies

Service life-cycle management policies focus on defining rules and regulations on business services. The appropriate SOA Governance policies have to be applied in each phase of the life-cycle to both guarantee and control the access of services. Figure 5.6 summarizes several policies that can be defined in each phase of the service life-cycle.

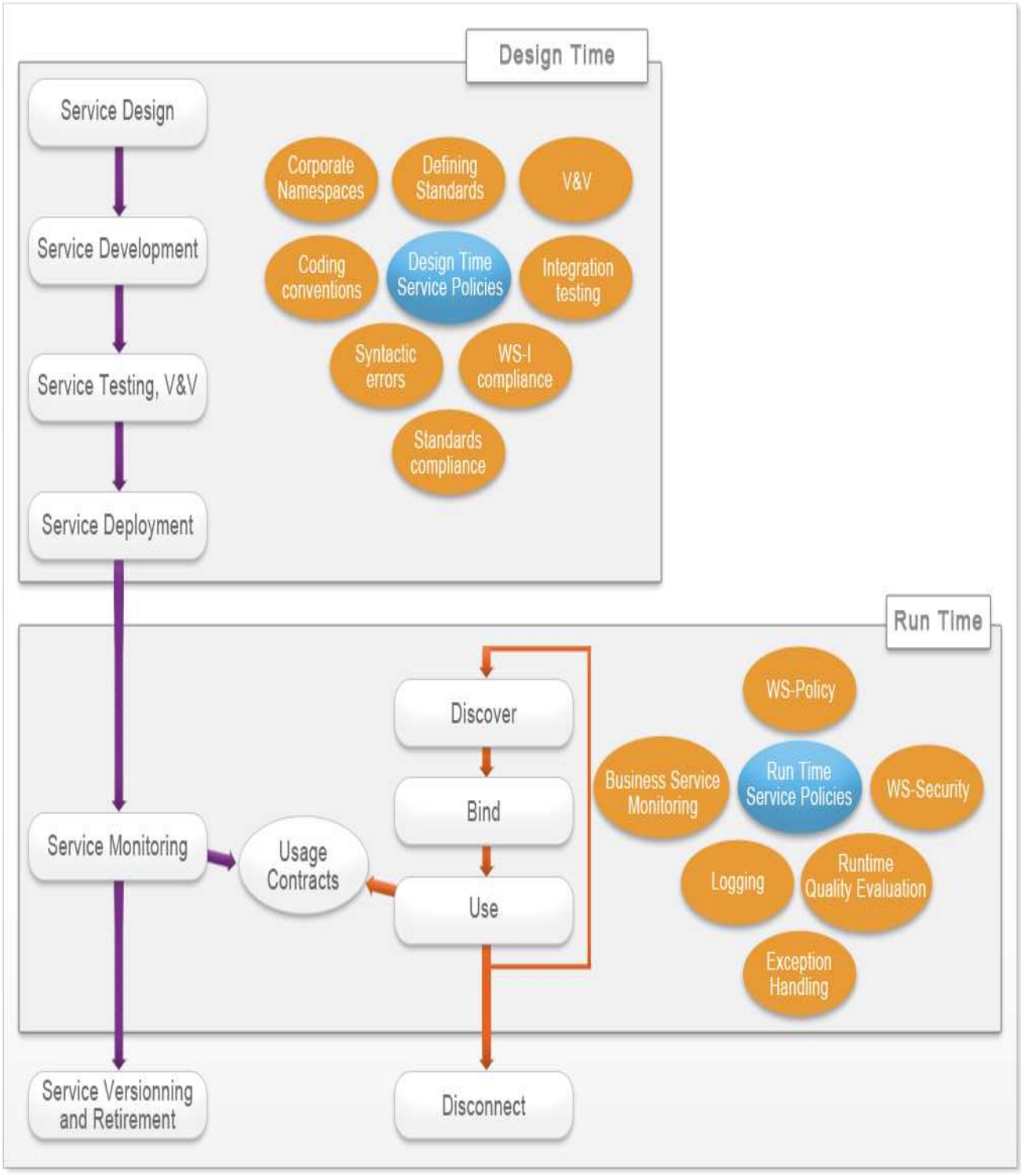

Figure 5.6: Service Life-Cycle Policies

Design Time Service Policies In a governance approach and at design time, each service passes several phases such as specification and design, development and implementation, testing and V&V before it is deployed.

The service design describes the infrastructure, capabilities, desired features and all the specifications about the service and the service candidates within a choreography. In this phase, a design time policy might define which, when, and where to use standards and insure compliance between them. The design time policies may also consider the fact of setting out corporate namespace, common coding conventions, identifying syntax errors, etc. Once the design of the service is finished, it is time to develop it respecting in one hand the rules defined in service design and in another hand, by defining appropriate policies such as coding conventions (for example, naming conventions for objects, variables, procedures,..) and syntactic errors policies.

Then, to assure the quality of services, service development needs to be coordinated with service testing with respect to integration testing and V&V policies. To support this coordination, the Rehearsal framework aims at providing features for applying TDD of services and choreographies. Using Rehearsal, the developer can first apply unit tests to validate atomic Web Services in isolation; thus, services can be combined with mocks or other services and integration tests are performed. Then, development-time conformance tests are applied to verify whether the services play correctly their roles in the choreography. Differently from the test strategies applied at run-time, at this point of development, the integration and the conformance tests are applied off-line in a development or testing environment.

Finally, policies at deployment stage might require that services in production environment are compliant with requirements of Web Service-Interoperability standard.

Run-Time Service Policies Policies at run-time stage come into play once the service is deployed and monitors operational aspects of a service to get full control of SOA Governance. It is most effective to define run-time policies in a Web Service Management (WSM) system to provide a common way to access and exchange information. Besides, run-time policies might require all deployed services to be managed and use the Web Service Security (WS-Security) standard and Web Service Policy (WS-Policy). Moreover, at run-time, it is ultimately necessary to enforce and manage the SLA contract elaborated between the service provider and the service consumer, according to defined policies.

5.2.2. Choreography Life-Cycle Management Policies

The governance policies that could be applied for the choreography during its life-cycle can be also divided as for the services life-cycle into two main classes: Design time choreography policies and Run-time choreography policies.

In Figure 5.7 we illustrate in a synthetic way the different policies that may be adopted in the CHOReOS governance framework.

Design Time Choreography Policies Design time policies include the respect of a set of policies relative to the stages of design of a choreography. First, to ensure its correctness a choreography needs to respect the structural and syntactic constraints dictated by the BPMN2.0 specification. For instance, a choreography needs to be triggered by a start event, end with end events, avoid infinite loops, etc. This kind of policies can be related to the syntactic formulation of a choreography. Second, a choreography must respect the enactment policies. For example, all services participating in a given choreography must be present (see Chapter 6). Other policies as for example the choreography ranking are described in the same chapter. Finally, a choreography must respect the design time service policies as described in Section 5.2.1.

Before choreography enactment, the developer must verify the correctness of the choreography. The Rehearsal TDD framework provides features to support this process. In the development or testing environment, the integration among choreography participants can be validated. In addition, the chore-

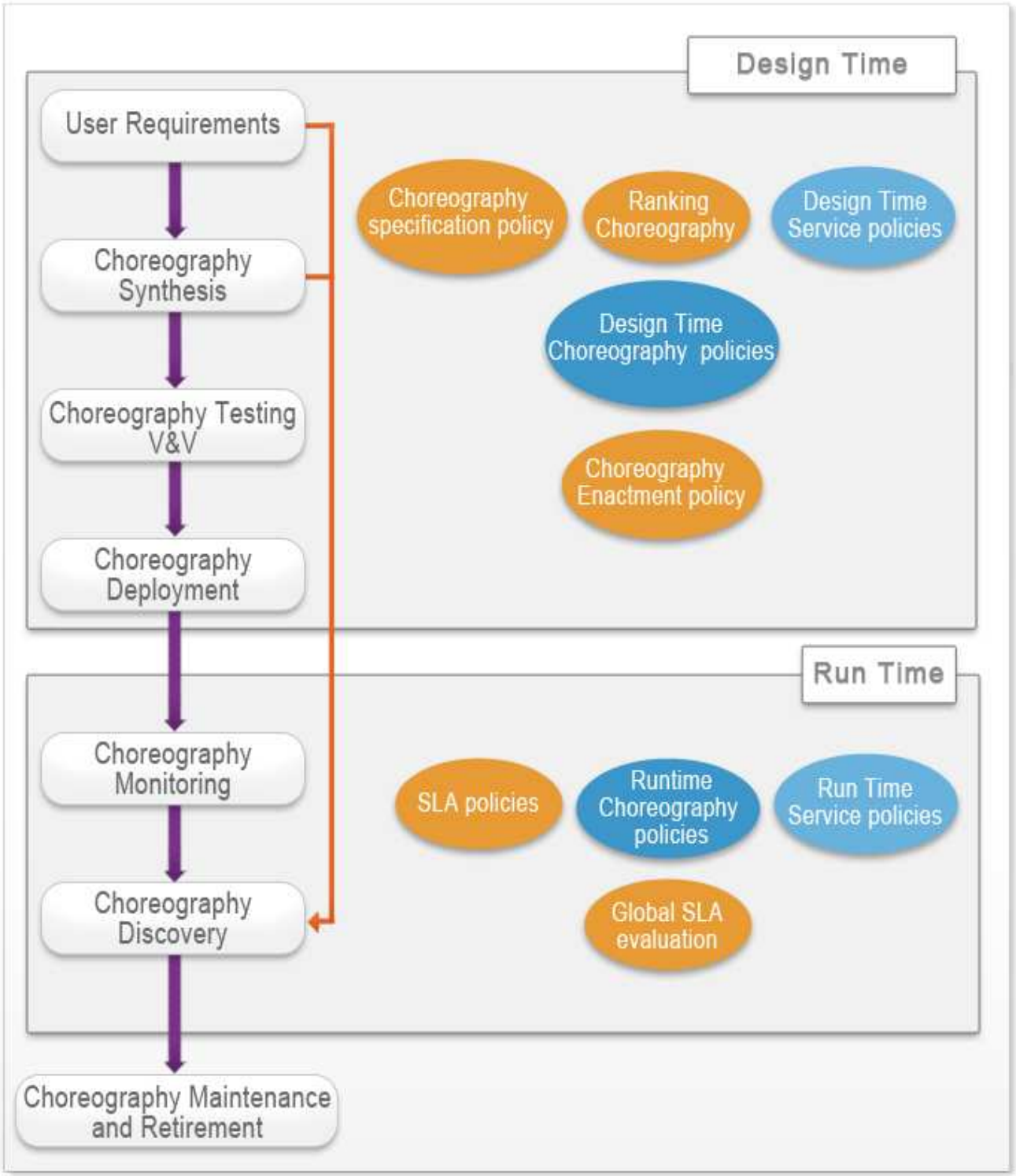

Figure 5.7: Choreography Life-Cycle Policies

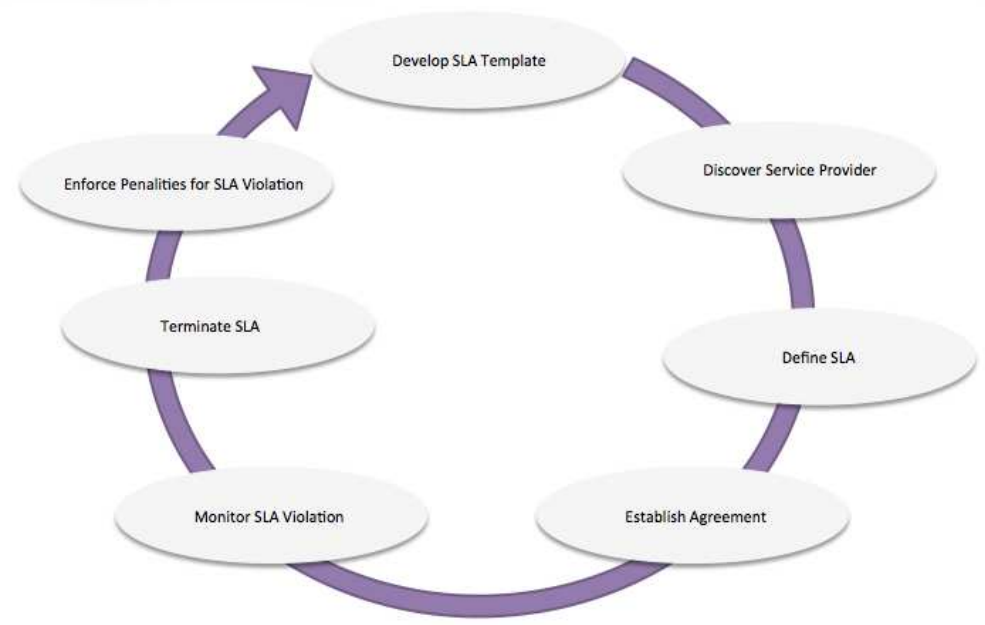

Figure 5.8: Service Level Agreement Life-Cycle [57]

ography functionalities can be validated using acceptance tests. Finally, scalability also needs to be assessed to assure that the choreography is able to support different load scales in multiple execution scenarios.

Run-Time Choreography Policies At run-time a choreography must respect the following policies. First, it must fulfill the non functional contract that was agreed. This non functional contract is based on the aggregation of several service level agreements that must be also respected at run-time by the services that are involved in the choreography. Second, a choreography must also respect the run-time service policies that are described in Section 5.6.

5.2.3. SLA Life-Cycle Management Policies

The Service Level Agreements are a commonly used way for designing non functional objectives to be reached by the business service once deployed. They are also needed for defining the responsible entities to be alerted if a constraint is violated. Besides, the service level agreement are useful in order to achieve the run-time quality evaluation of running business services. They are used later in order to calculate the choreography level agreement. In this section we present the SLA life-cycle as we will consider in the CHOReOS governance framework.

The SLA life-cycle [57] includes the states of discovery, negotiation, creation, monitoring, termination, and enforcing consequences for non compliance. Figure 5.8 provides an overview of the following SLA life-cycle stages as defined by Sun Microsystems:

1) Develop SLA Template: the whole life-cycle of the SLA is based on a template or model used for the representation of its various clauses. This first phase consists in the specification and modeling of the template,

2) Discover Service Provider: the SLA life-cycle starts with discovering resources that could satisfy the requirements of the service consumer.

3) Define SLA: once the service provider(s) has(ve) been discovered, it is important to specify and model the required quality levels of the agreement.

4) Establish Agreement: in this phase the SLA template is created and is accomplished by signing a service level agreement by the client and the provider.

5) Monitor SLA Violation: this phase consists in monitoring the obligations defined in the SLA to ensure that all contract clauses have been achieved or violated by one of the parties or both of them.

6) Terminate SLA: This phase consists on the termination of the SLA.

7) Enforce Penalties for SLA Violation: In case of non compliance, it is important to specify the consequences and penalties.

5.3. Non-Functional Governance Policies

Web Services are usually designed and developed in order to primarily meet a set of functional requirements. However, for an efficient participation in a corporate environment, possibly in an integration with other web services, a web service should also provide some guarantees about its efficiency, availability, dependability, and so on; in short, about its quality of service (QoS). Consumers (and partners in integrated environments) will often expect to be given such guarantees, and a consumer's choice between different service providing the same functionality may well be based upon the guarantees that they provide. For this reason, maintaining high QoS guarantees may be key to a web service's success against its competitors.

The importance of a service's QoS is not limited to the moment of the choice, but rather through all the life-cycle of the service, because if the initial guarantees were to fail, the service might lose its trust among its consumers. To achieve this, the QoS does not only need to be provided and maintained once the service is built, but it should also be monitored at all times, to ensure that contingent changes do not cause a quality loss.

In general, a service composition whether an orchestration or a choreography, will expect its composing services to provide a certain degree of guarantees, a minimum level of QoS that all service must comply to. On one side, this means that a service, to be included into the composition, will need to provide and maintain the minimum required QoS requested by the composition; on the other side, the composition as a whole will have to verify the compliance of the service with the required degree of QoS, both at the time of the admittance and with a constant monitoring of the service provided. In the case of an orchestration, the latter depends solely on the orchestrator, while in a choreography there is no centralized hub, so a sort of policy agreement between the participants is needed.

5.3.1. SLAs Policies for Choreography

A choreography describes a collection of web services that collaborate together to achieve a common purpose. It captures both functional and non functional interactions in which the participating services engage to achieve this goal. To guarantee the non functional part the choreography, it would be necessary to define a global SLA for this choreography. Indeed, it is a form of policy that the choreography must respect in order to satisfy the user requirements.

In CHOReOS, we need to provide a governance solution that considers a choreography level agreement between a very large collection of services. We may inspire from research works such as presented previously and adapt them to FI requirements. We may further investigate research directions dealing with a very important size of the service collaborations within a choreography. Besides, the complexity of the collaboration graph may vary according the complexity of the enacted choreography of services. Consequently, several choreography level agreement generation strategies could be studied.

In CHOReOS we will investigate the implementation of choreography-level SLA approaches in both top-down and bottom-up directions. In top-down way, our idea is to introduce a notation and a process

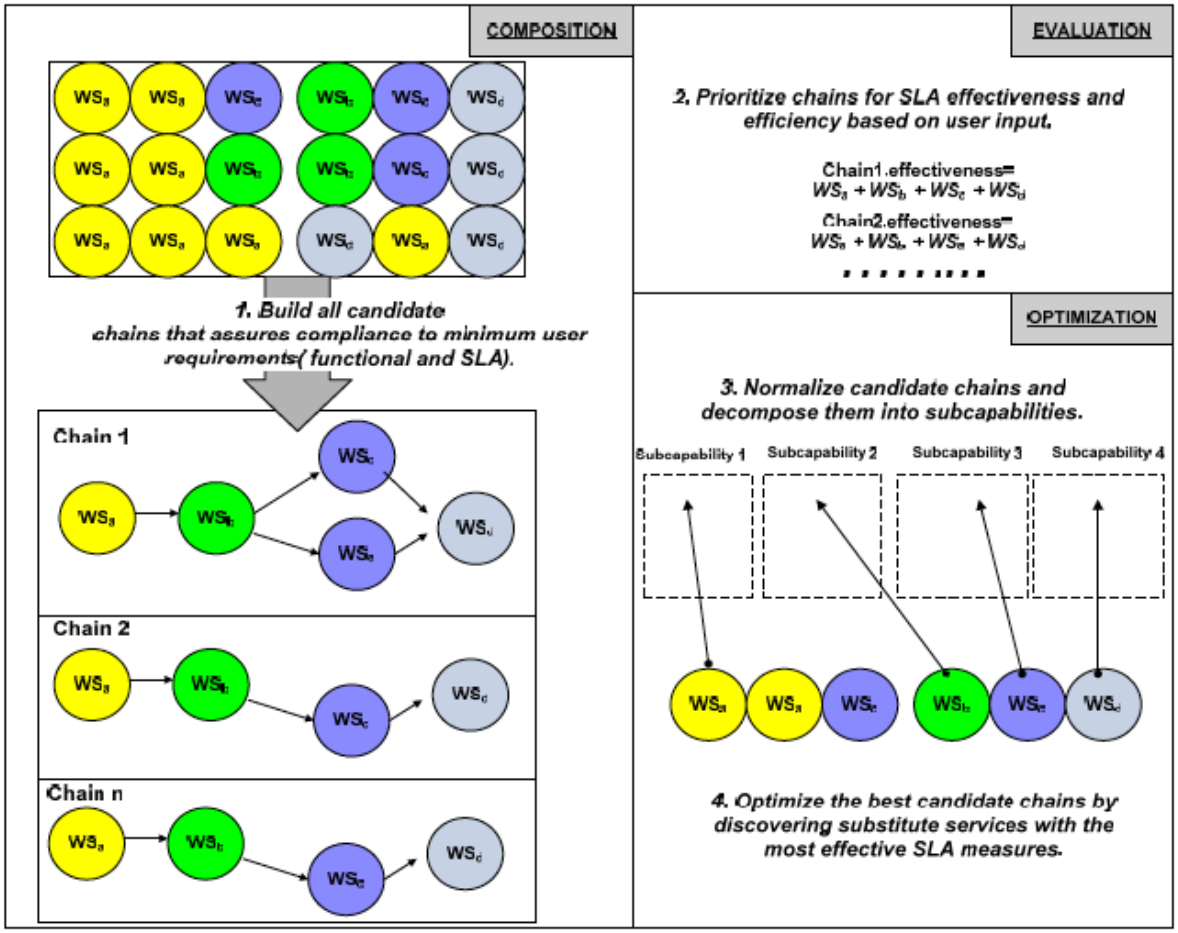

Figure 5.9: The SLAs Approach as Described in [24]

for augmenting a choreography specification in BPMN [1] with non-functional requirements. This will be discussed in the next section. In bottom-up way, we need to define a choreography level agreement based on the composition of multiple service level agreements. The use of SLAs is explored in many related projects and in several domains. In the following, we present two research approaches that are close to the problematic of choreography level agreement in CHOReOS.

Blake and Cummings [24] introduce three aspects of SLAs composition which are Compliance, Sustainability, and Resiliency. Compliance or suitability ensures that the consumer receives the response expected at the service level that is required. Sustainability is the ability to maintain the underlying services in a timely fashion. Finally, the resiliency aspect is the principle of a service to perform at high service levels over an extended period of time. The authors define a methodology for web service composition combined with workflow-based SLA composition. This process is divided into three parts which are : composition, evaluation, and optimization. The composition procedure consists on building all candidate chains of web services using forward-chaining or backward-chaining that match both functional and SLA requirements. The evaluation step consists in prioritizing the multiple chains generated in the first step using the user priorities. The third step is the optimization and generation of the best service chain. This process is illustrated in Figure 5.9.

Haq and coauthors [34] define a view based formal model to describe hierarchical service level agreement in supply chain scenarios. Their approach is in compliance with the WS-Agreement standard and consists of generating an aggregated SLA-choreography from the different SLAs. The approach studies the several dependencies between SLAs.

[1] Note that in this document, the acronym BPMN always refers to version 2.0 of the specification

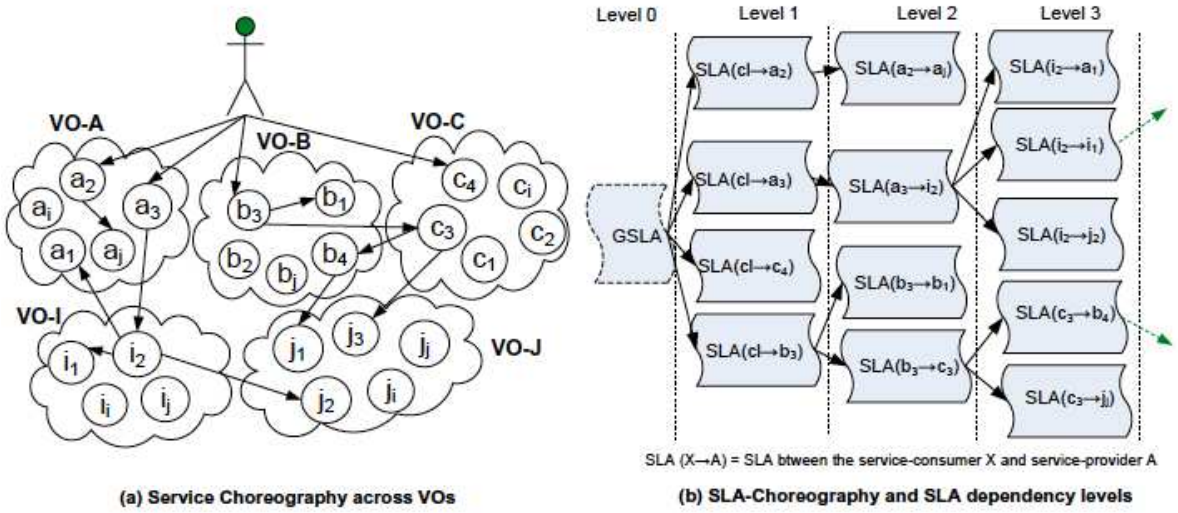

Figure 5.10: The SLAs Approach as Presented in [34]

Indeed and as presented in the Figure 5.10, the client is connected to some services that collaborate with other services in order to achieve the common goal. Each pair of services establishes an SLA contract in which they define their requirements.

To generate the SLA choreography (GSLA), the authors decompose the service chains resulting from the coordination into several levels taking into account the dependencies between them. The aggregation of these SLAs gives rise to an SLA-Choreography in connection with the underlying service choreography.

The CHOReOS project evolves in the FI environment where a very large number of services interoperate. Insuring a global level agreement for choreographies relies on the composition of an important number of SLAs contracted between services involved int the choreography. We consider both works presented above, and especially the research work provided by Haq and coauthors, as very close to our problematic. Nevertheless, they don't focus specifically on composing SLAs in a very large scale.

5.3.2. Specifying Choreography-level NF Requirements

Behind any approach taken for decomposing and evaluating choreography-level SLAs, we also need to devise a notation by which a choreography designer can express them. We are considering to introduce non-functional annotations at the choreography level. In particular, we need to augment the BPMN choreography specification language with extensions apt to deal with quality of services aspects. Hence, we are currently investigating appropriate SLA notation and a metamodel behind for BPMN.

Indeed, choreographies define the expected behaviour of the interacting services, or "Participants" in BPMN terminology. The OMG specification document for BPMN states that basically a choreography establishes a "procedural contract" ([1], pag. 25). The standard BPMN definition however currently only refers to functional aspects for such hypothetical contracts.

On the other hand, given the increasing importance for non-functional aspects of service-based systems, choreographies should also define constraints on QoS and other non-functional properties which participant services should comply with. *Our intuitive idea is that the choreography specification should also establish non-functional contractual agreements, or SLAs, at the choreography level.* Such agreements would entail not a specific concrete service, but rather the abstract description of participants. In other words, SLAs attached to a choreography description would postulate the expected quality levels to be guaranteed between participants playing the choreography roles. As intuitive and simple as such an idea is, we found no proposal for it or a similar approach in the literature.

In the next deliverables we will concretely implement these ideas as follows Since our aim is to express – together with the specification of the expected functional behaviour of choreography participants – the expected and/or required non-functional properties, we will refer in this stage to the BPMN *Choreography Diagram*, with the purpose of modeling the non-functional (NF in the following) properties and related constraints, to which services which enter the choreography will have to abide by.

We will thus specify graphical annotations to the *Choreography Diagram*, for expressing the required NF properties and corresponding constraints. The goal of such annotations is to provide the reference model against which the enacted choreographies and the concrete services playing the specified participants roles will have to be assessed.

For supporting such conformance analysis, we will also need to be able to process and decompose such annotations at the choreography level down to their atomic activities, and for each activity to the composing events. Our idea to achieve this goal is to map the annotations in the *Choreography Diagram*, which will be expressed at an abstract level, either onto the UML Profile MARTE [50], or onto a *Property Meta Model* (PMM) which we recently developed. The latter is a generic conceptual model which defines non-functional properties related to performance, dependability and security, and expresses them as operations between events. In the following we will only report some main concepts of PMM which are necessary to make the description self-contained, and refer to the existing literature for a detailed description. In particular, an outline of PMM can be found in [27], whereas the complete definition is made available on line, and is currently continuously updated at `http://labse.isti.cnr.it/tools/cpmm`.

We are also exploring the possibility to adopt a model-driven approach for translating the proposed BPMN NF annotations to the input notations of the CHOReOS IDRE components, in particular the on-line monitor. To this purpose we need a way to deal with the complexity of transformation towards several possible differing notations. Our plan is to pass through an intermediate model (the “kernel”) by pruning the information from the source model that is not needed to be included in the target model, while still retaining all needed one. Among the transformation approaches that make use of intermediate models in the area of NF attributes analysis, Petriu et al. [46] proposed the core scenario model (CSM), and Grassi et al. proposed KLAPER in [33]. CSM is specifically related to performance analysis, while KLAPER is intended to also serve for reliability purposes. Both CSM and KLAPER are defined as a MOF (Meta-Object Facility) metamodels, so the standard MDA-MOF facilities can be used to devise rule-based transformations from CSM/KLAPER models onto elements of the target metamodel. However, the detailed formal definition of the BPMN NF annotations and their translation to the input notations of the CHOReOS IDRE components is outside the scope of this document, and will be addressed in the next CHOReOS WP4 deliverables.

In the *Choreography Diagram*, the interactions between participants happen through the exchange of *Messages*. The atomic activity of a BPMN choreography is the *Choreography Task*. A *Choreography Task* is identified by a task name, and has two or more participants, which may also be associated with message flows. Concerning the *Choreography Task*, this could be annotated with NF required properties which apply to all task participants, hence these will be cumulative properties. The annotation to a *Choreography Task* consists of a property, and is optional.

The portion of the PMM meta-model defining a Property is reported in Figure 5.11. With reference to PMM, a Property has two mandatory attributes, which are “Nature” and “Class”. In general, properties in PMM can be of the “Abstract”, “Descriptive”, or “Prescriptive” Nature, where an Abstract property indicates a generic property that does not specify a required or guaranteed value for an observable or measurable feature of a system, and a Descriptive property states that the given participant is able to provide that guarantee. Specifically, the annotation to a *Choreography Task* will be of Nature “Prescriptive”, i.e., it indicates a property that a partner requires in order for others to participate in that interaction.

In general, the Class attribute in PMM can take the following values: Performance, Security, Dependability and Trust. At present, we do not consider Trust-related properties. Hence, the annotation for a prescription relative to a Choreography Task will refer to:

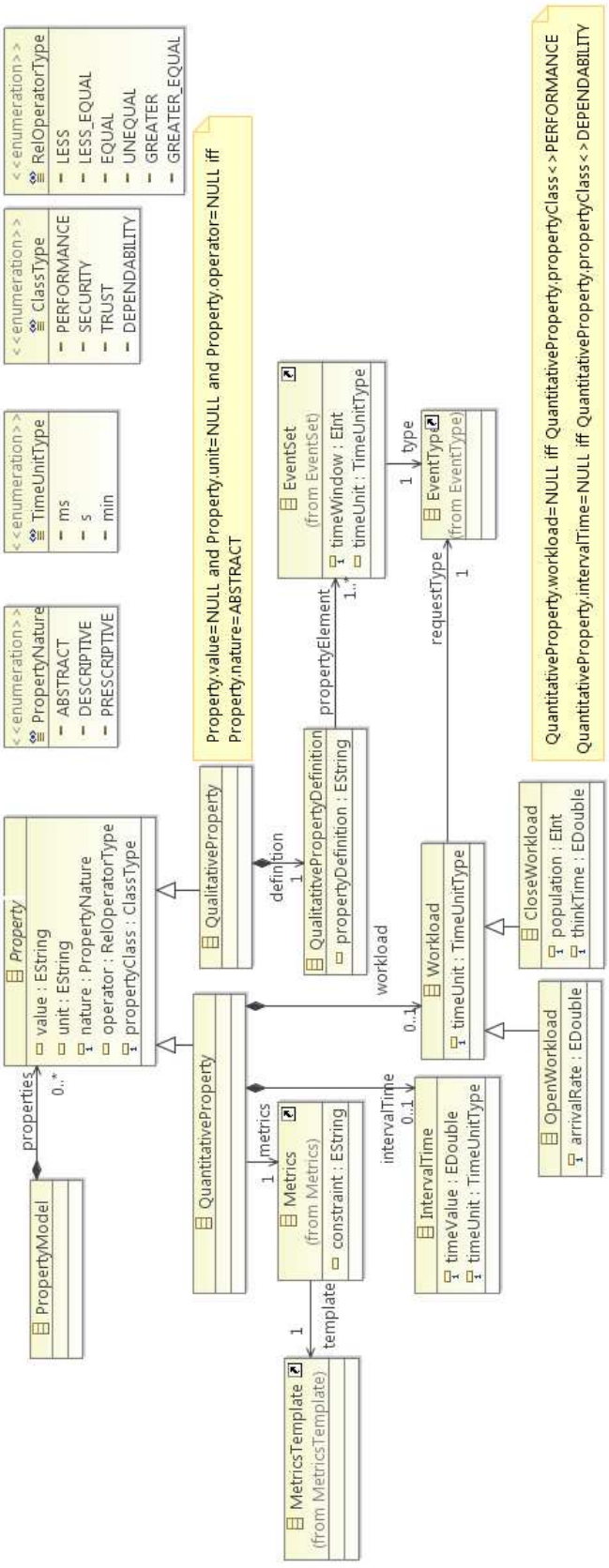

Figure 5.11: Property Conceptual Definition

Performance: either a max duration (latency), an absolute validity expiration date for a task, or a required protocol designed for performance (e.g., data compression), or a minimum number of allowable simultaneous connections;

Security: a security level for the task, or a required protocol for encryption;

Dependability: a max failure probability, a required availability value, or a required protocol for fault tolerance (e.g., supporting retransmission of failed packages).

Note that a single task can include more than one property annotations, belonging to one or several different Classes. It is the choreography designer's responsibility to ensure that the annotations are satisfiable (or realistic) and not mutually incompatible.

Due to the ULS dimension, when prescribing some NF properties, it is important to make explicit if these are to be taken as "Hard" or rather as "Soft" constraints, in the usual meaning of real-time computing. Hard properties are required to be guaranteed to consider a solution valid, whereas soft constraints are not and provide a way to express preferences between different solutions. Clearly when dealing with a huge number of services and transactions, ensuring either Hard or Soft constraints may make a huge difference in terms of costs. We adopt a different notation for Hard and Soft properties: the former will be accompanied by an exclamation mark in the graphical view.

In BPMN, *Choreography Task*s can have markers that denote how the task may be repeated. To make things simpler, the NF annotation in such a case will always have to be interpreted as relative to every single repetition task. In such a case, the resulting NF property will vary according to standard composition rules (e.g., [51, 25]). In the following we explicitly describe these rules when the Class type is Performance.

- If the Choreography Task includes a standard *loop* marker, t is the maximum latency allowed for the task, and the loop is executed n times, the global resulting NF property annotation will be the maximum allowed latency equal to $n * t$.
- if the Choreography Task includes a *parallel multi-instance* marker, t is the maximum latency allowed for the task, the global resulting NF property annotation will be the maximum allowed latency equal to t.
- if the Choreography Task includes a *sequential multi-instance* marker, t is the maximum latency allowed for the task, and the sequence length is k, the global resulting NF property annotation will be the maximum allowed latency equal to $k * t$.

Such annotations can also be recursively used for Sub-Choreographies, in those cases where they might be referred to a set of tasks. In this case we will introduce a Choreography Sub-Process notation, and will put the NF annotation in correspondence of the Sub-Process name. This means that the prescribed NF property applies to the included Sub-Choreography as a whole. Thus, if the annotation corresponds to a Performance property the duration or the expiration date refers to the whole Sub-Choreography; if it gives a security level, this must be guaranteed for each single task within the Sub-Choreography, and finally if the annotation corresponds to a max failure probability or a minimum availability, these must be guaranteed as a whole by the process.

If the annotation is meant as global for a whole Choreography Diagram, then we will annotate the Start event (denoted by a circle) with the prescribed NF property.

The annotation of NF properties can in alternative be put within a participant band, as an attribute to the participant name. In such a case it is meant as a prescription on the concrete services which will take the corresponding participant role. The distinction between Hard and Soft property applies also to participant annotations.

Figure 5.12 presents an example of the annotations described above. The example is a very simple choreography consisting of two sequential tasks. In the figure, all of the above concepts have been

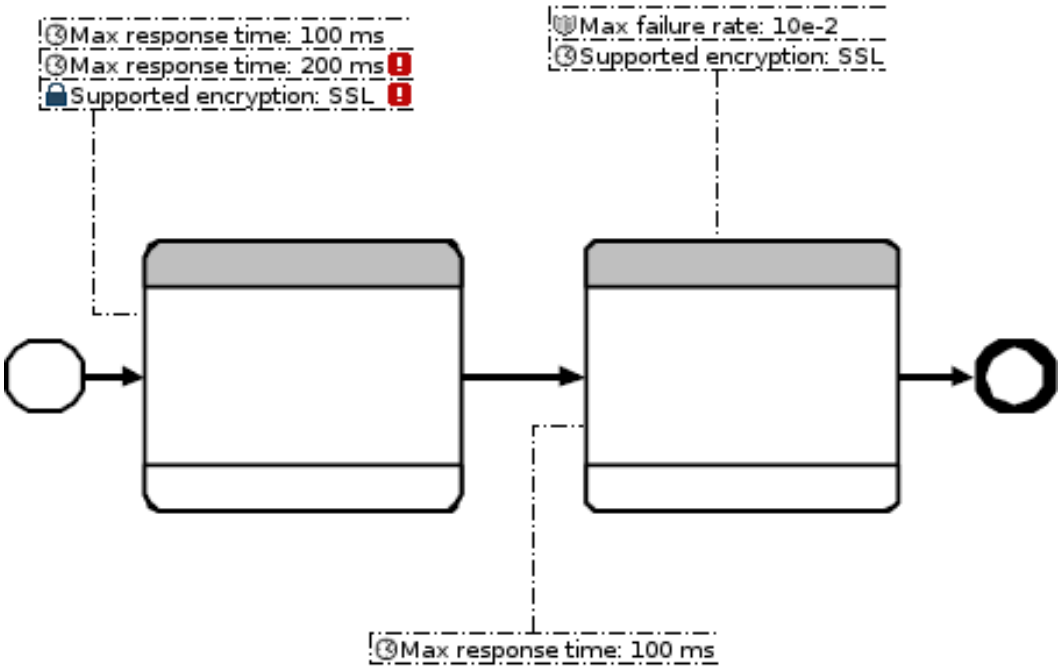

Figure 5.12: Example of a NF Choreography Annotation

shown. The first task has three NF requirements; two are mandatory ("hard"), while one is optional ("soft"). The first task must necessarily be executed within 200 milliseconds from its start (hard), and preferably within 100 milliseconds. This means that if two different services providing the same functionality respond in, e.g., 80 and 90 milliseconds respectively, they will both be equally adequate to participate in the choreography, but if they respond in 80 and 110 milliseconds respectively, the former one will be preferred over the latter, because it satisfies a soft requirement the other one can't satisfy. Additionally, the first task necessarily requires that the interaction be encrypted with SSL.

The second task has a soft requirement on response time, which should be preferably no more than 100 milliseconds. However, specific requirements are also expressed for the non-initiating participant (the grey band), which would ideally be a service able to satisfy those two requirements (although both are soft). First off, it should guarantee an error rate of no more than one erroneous response every 100 invocations; secondly, it should be able to support SSL encryption (the reason for this might be that clients would trust the service more if they were able to send encrypted data), although the functionality would still be accepted over a non-secure channel.

5.3.3. Run-Time Quality Evaluation

The scope of SOA Governance is not limited to deploying and making services available to potential consumers. As we said previously, SOA Governance covers the entire life-cycle of services, including the run-time stage.

Specifically, the governance at run-time also deals with the monitoring functionality. The business service monitoring component is responsible for monitoring both communication and non functional concerns. It also focuses on the hardware resource monitoring. In this section we address the non functional monitoring of services, the Run-Time Quality Evaluation. It consists on the control, supervision, and evaluation of the SLAs contracts prescribed by services within an enacted choreography.

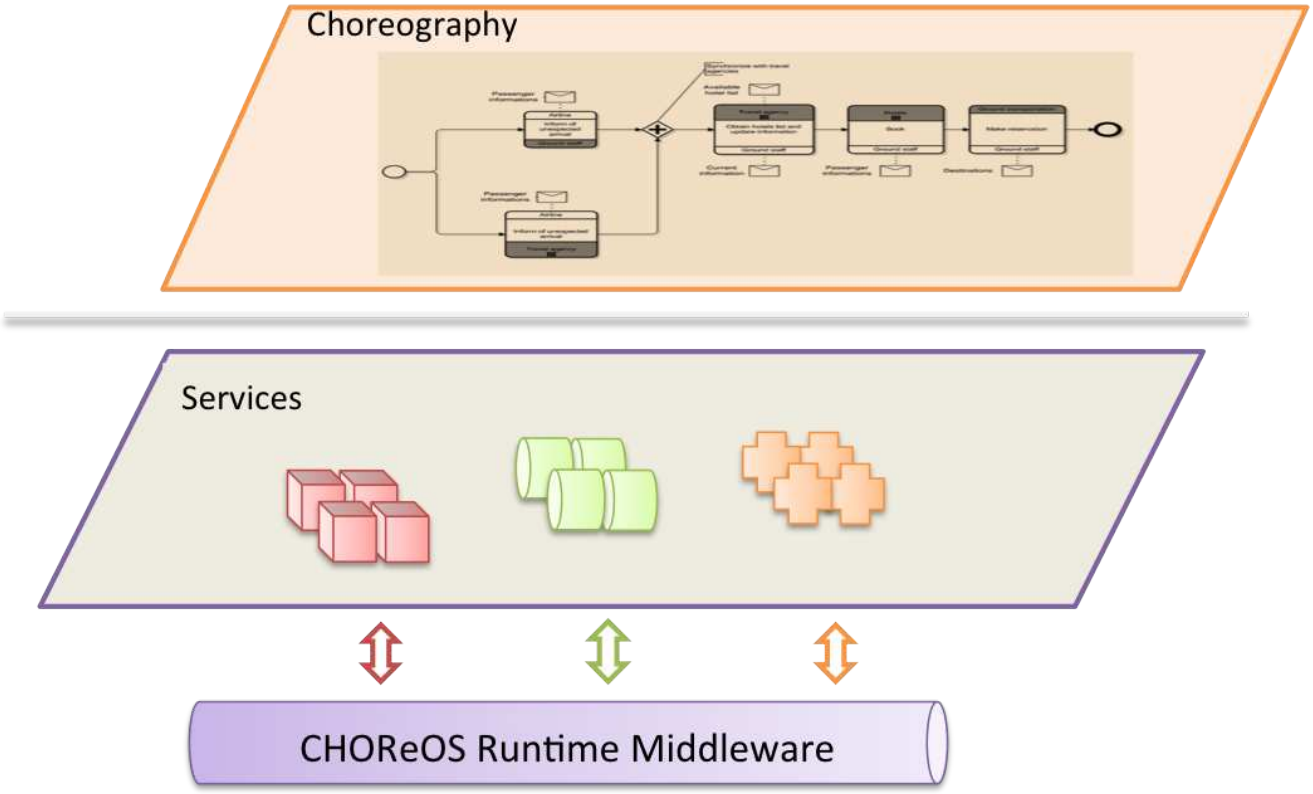

Figure 5.13: Layered View for the Choreography of Services

Noteworthily that the evaluation of either a policy or a SLA is a complex task that usually goes beyond the mere technological solution, involving also legal, or social aspects. Furthermore, an effective enforcement activity is usually domain-specific, and it is strictly related to the contexts where it is applied.

Specifically, the quality evaluations on a choreography are realized by making sure that the interactions between its several partners happen according to the usage contracts previously agreed. A top down approach is adopted to monitor, to check the SLA contract, and to evaluate the communications regarding to the choreography specification and the corresponding services (see Figure 5.13). The run-time quality evaluation of the choreography and its services will be realized as described in the following.

The monitoring phase (logging,auditing) of the run-time governance, may be clarified examining the monitoring infrastructure that will be used into the CHOReOS governance framework.

Monitoring has been defined as *the process of dynamic collection, interpretation, and presentation of information concerning objects or software processes under scrutiny* [39].

Logging is the task in the policy life-cycle that collects statistics to support the process of checking for policy compliance.

The monitoring component is involved into different phases of the execution of a choreography like data collection, interpretation and presentation of information concerning the “observable object”. Some details about the architecture and the module that composed the monitoring have been anticipated in Section 3.2.

The most commonly used approach for observing the behavior of distributed systems is event-based monitoring. We will develop and use an event-based monitoring infrastructure, that implements the key components of a generic, flexible and robust architecture for composite event detection by means of publish/subscribe messaging pattern.

Elaborating on [43], five core functions can be identified for such a generic monitoring system (see Figure 5.14):

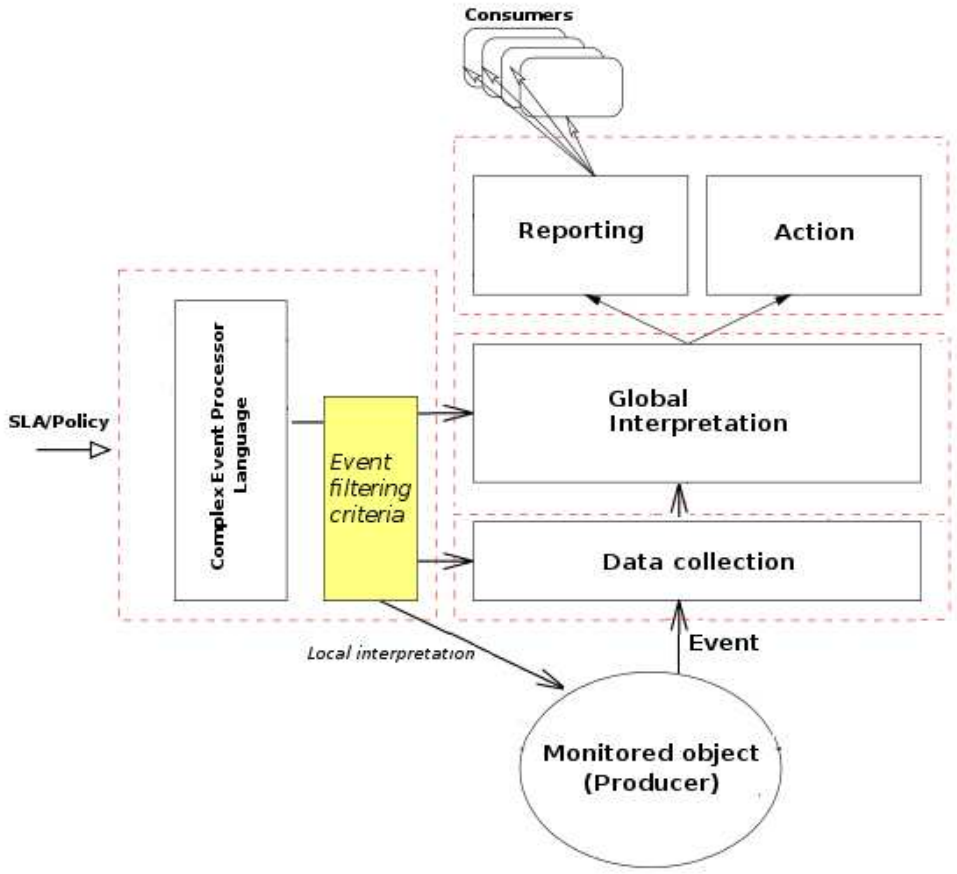

Figure 5.14: Generic Monitoring and Run-Time Quality Evaluation Infrastructure

1) Data collection: this function concentrates on the collection of raw data from the execution of the observed components. This can be done by either instrumenting the subject component (when this is possible) or by intercepting interactions among components through a proxy-based probe. A special case is represented by the built-in logging facilities that many systems provide natively.

2) Local interpretation: this function refers to the filtering that raw data go through before being interpreted at an aggregated level, to remove redundant or irrelevant information.

3) Data transmission: in distributed systems, this function takes care of gathering information from different originating nodes to a central (possibly not unique) node. Data transmission might exploit smart optimization algorithms (e.g., to delay data transmission or to give higher priority to certain information, when the network is subject to congestion).

4) Global interpretation: (also known as “correlation” or “aggregation”), this function makes sense of pieces of information that come from several nodes and puts them together in order to identify interesting conditions/events that can be observed only at an aggregated level. This function can be realized by means of a complex-event processing engine.

5) Reporting: this function deals with presenting the results of monitoring in a format that is meaningful to the “consumer” of the monitoring system. The consumer can be a human (e.g., a system administrator) or a program (e.g., a software component that implements the feed-back loop in a self-controlled software system).

The CHOReOS monitoring subsystem will cover these five functions in a modular and flexible way.

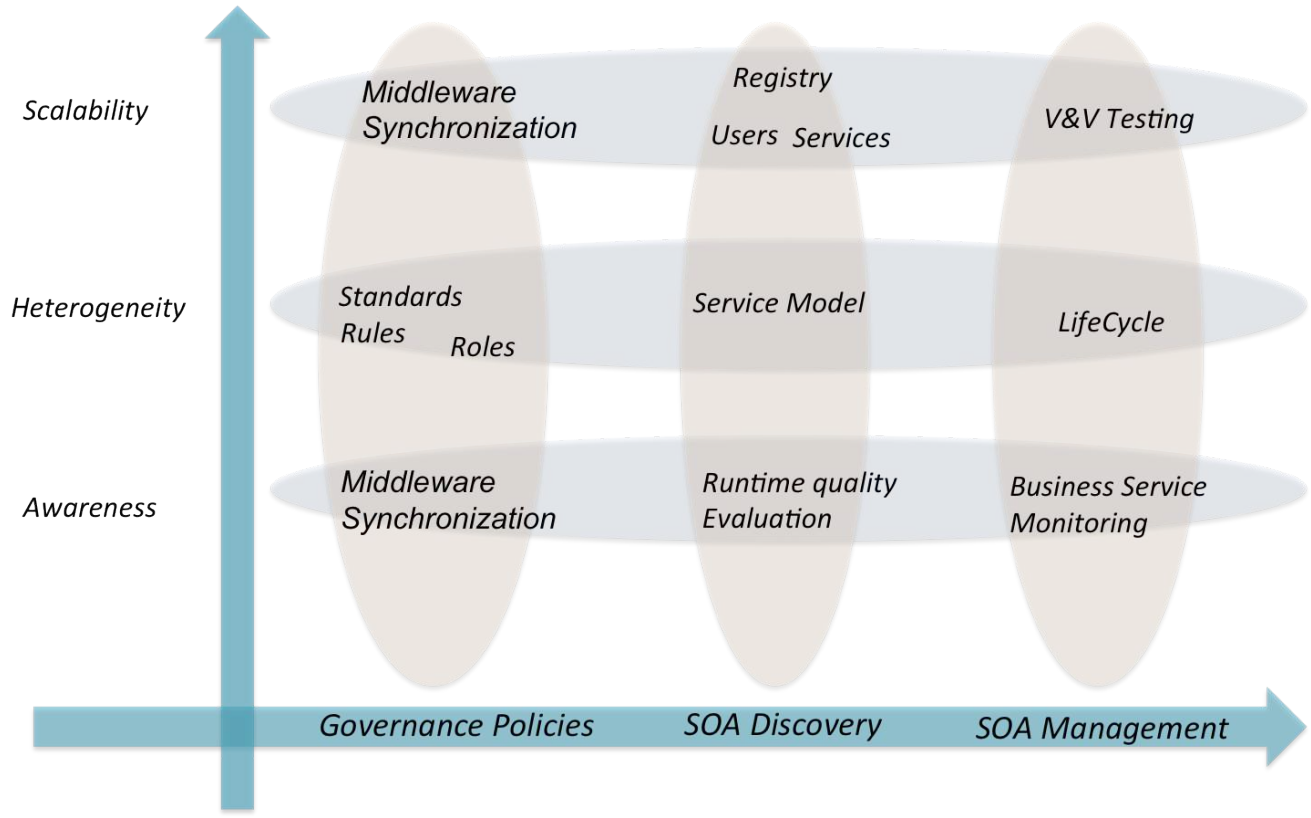

Figure 5.15: CHOReOS Governance Framework Responses to FI Challenges

5.4. CHOReOS Governance Framework Responses to FI Challenges

In Section 2.1 we have presented the main concepts behind SOA Governance and identified the Governance Policies, the SOA Discovery and the SOA Management. In the ULS FI context, the CHOReOS Governance Framework will face a significant number of services, users, choreographies and policies. In Figure 5.15, we projected the main domains close to SOA Governance(Governance Policies, SOA Discovery and SOA Governance) with regard to the FI dimensions of scalability, heterogeneity and awareness. Below, we discuss the responses provided or planned by the CHOReOS Governance Framework to the respective challenges:

Scalability The scalability dimensions may vary according to the number of services, consumers and requests. The CHOReOS Governance Framework needs to be able to tackle such issues.

First, the governance registry is dedicated to storing the services descriptions and the Service Level Agreements needs to be able to support a large quantity of data. A preliminary solution is already provided by the Extensible Service Discovery Service proposed in WP2 which offers storage mechanisms able to host a significant number of services descriptions and service level agreements. The governance registry will augment such functionality by providing a dedicated storage mechanism supporting the relevant data concerning services. This way a multi-protocol storage mechanism is ensured making the CHOReOS IDRE capable of dealing with scalable dimensions.

Second, we make the choice of synchronizing the governance registry with the CHOReOS run-time middleware in order to fill the gap between the design and run-time environments. This provides a run-time registry for the available services in the environment. The run-time middleware for business services is provided by the distributed service bus. The latter will be adapted for ultra large scale dimensions and a deployment on top of the cloud. These ULS enhancements will

impact the governance registry as it is synchronized with it.

Third, the V&V and TDD capabilities will help testing services and choreographies for ULS support. Further works, need to be undergone in order to test the ability of the governance framework for supporting an important number of services and requests in test bed and real conditions. Finally, we will design the Governance Framework so that it fits to a distributed and scalable environments. Modular and flexible architectures will be adopted.

Heterogeneity Business Services handled by the CHOReOS Governance Framework have heterogeneous sources. Actually, some business services are developed using the governance framework and thus tested, Verified and Validated. Others are discovered at run-time and can be developed in a different way. The scalability dimension combined with the heterogeneity decouple the complexity of managing heterogeneous business services. Heterogeneity can be expressed at several levels of the business service.

First, the business service description can be made in different specifications and languages. Providing a governance registry that enables the discovery and query of business services, relies on the extraction of the relevant data from several different service models. The CHOReOS Governance Framework answers this issue by adopting a uniform service model based on USDL unified Service Description Language. This business service model will bring a uniform abstractions for the several discovered services.

Second, the multiplicity of the users interacting with the CHOReOS Governance Framework is answered by the identification of the most relevant roles and the specification of the relative use cases. This work is presented in Section 5.1.

Third, policies described by business services may be very heterogeneous according to the services uses. Business services involved in a choreography may emphasize their need for the enforcement of security concerns, others in a less critical context may not. Finally, the CHOReOS governance Framework will tackle to such issues by adopting strategies and rules that will regulate the management of policies. This contribution will be the subject of the deliverable D4.2.

Awareness In a context continuously evolving with the appearance of new services and new users, the awareness of the external environment is essential. The CHOReOS Governance Framework achieves awareness by the synchronization with the run-time middleware.

First, in order to get the knowledge of the running service on top of the CHOReOS Run-Time Middleware, the Governance Registry is synchronized with the extensible Service Access Middleware and more specifically the Distributed Service Bus. The endpoints of the services available in the run-time environment are imported and governed at run-time. This synchronization will be supported by the CHOReOS Governance Framework by the implementation of the needed components for connecting the registry with the middleware.

Second, the CHOReOS Governance Framework keeps informed about the run-time behavior of the running business services thanks to the business service monitoring. The latter monitors at run-time at several levels: communications between services, service level agreements, and hardware resources. The monitoring augment the awareness of the Governance Framework and reduces the gap between the design time and run-time environments. Consequently, according to the monitored data suitable decisions can be taken promptly.

The CHOReOS Governance Framework need to answer FI challenges at several levels. Preliminary choices are made in this deliverable in order to take into account the scalability dimensions. Further considerations regarding the framework design and construction will be adopted.

6 Governance Policies and Rules enabling V&V Activities

As described in the previous chapters, CHOReOS foreseens that part of the V&V process takes place at run-time. Thus, the V&V framework includes rules, policies, and governance mechanisms supporting the on-line validation of both choreographies, and the services they refer to.

In the following we propose and discuss an initial set of policies and rules we are developing within the V&V governance framework of CHOReOS. In particular, for clarity of exposition we organize such policies according to the following classification:

V&V Activation Policies that describe rules for regulating the activation of the on-line testing sessions. As described in the following, such activation could be either periodic, driven from the events about the life-cycle of a service, or linked to some kind of quality indicator;

V&V Rating Policies that prescribe which aspects have to be considered for rating the quality of both choreographies, and services.

V&V Ranking Policies defining the rules and the metrics for computing the quality parameters expressed into V&V Rating Policies;

Choreographies Enactment Policies that prescribe rules for decides when a service choreography could be enacted;

Test Cases Selection Policies regulating the testing strategies that can be applied at run-time within the Governance V&V Framework;

Ultra-Large Scale Policies that regulates V&V Activities mitigating the challenges introduced by ULS dimension .

The rest of the chapter introduces such policies and rules of the Governance V&V Framework by considering a given choreography C.

6.1. V&V Activation Policies

Originally [23], the very general idea supporting V&V governance was to introduce an on-line testing phase when a service asks for registration within a registry.

In this vision, only services that passed the validation steps foreseen by a the governance framework are guaranteed to be listed in the registry. As a result, a registry supporting an on-line testing phase, is expected to include only “high-quality” services.

In addition to the registration of a new service, the on-line validation process could be also extended to others events like the release of a new service version.

Note that, if on the one hand the new registration of a service on the registry could be considered as an explicit activity by a service provider that is willing to promote its service; on the other hand the notification of a service upgrade could be sporadically notified. The governance mechanisms described

in Chapter 5 could mitigate this aspect by means of specific policies, and obligations that the service providers should abide binding their services to any choreography in CHOReOS.

During the life-cycle of a C, a service that was originally registered to play a given role in C could be or result deprecated. This event may impact on the regular progress of some of the activities described in C. Thus, the V&V governance should support rules that verify if the service registry still points at one or more active services that could play all the roles subscribed by the removed service in C.

Note that, also in this case a service provider may omit to notify the deprecation of a service to the service registry. The monitoring system infrastructure of the V&V governance architecture could cooperate with the Service Registry in order to analyze and notify if any registered service is either not available, or not reachable anymore.

Within a choreography a service may play one or more roles. Also, the same service may be involved in several choreographies. Service provider may decide to change the roles that their services are playing in the choreographies in order to fulfill new needs, new business requirements, or the evolution of the technical solutions offered by the service. In this scenario one of the V&V governance rules we propose is that any modification (i.e. activation, modification, cancellation) to the role of a service in C should activate a new testing session. Specifically:

activation : when a new role A is added to a service S in C, thus we could execute the integration test suite for A in C and run it through S. Evaluate the impact of the result of such on-line testing session of the testing scores of the choreography C.

cancellation : when a role A is deleted from the roles that a service S in could play in C, verify that the service registry still points to at least one active service that could play A in C.

modification : this step could be processed as a sequence of a cancellation and an activation of a role for a service in C.

In addition to the event-driven activation of the on-line testing sessions (e.g. service registration, etc), the V&V Governance infrastructure should support the “perpetual” testing of software services, by also including test schedulers that activate on-line testing sessions either periodically, or when a specified deadline is met.

V&V governance is part of a continuous process aiming at specifying and at supporting the enforcement of decision with respect to model-based testing approaches. Specifically, such approaches could be applied both off-line and on-line during the whole life-cycle of a service choreography. Thus, a dynamic validation process for ULS choreographies likely foresees that tighter or better (i.e. either more efficient, or more effective) test suites could be defined for choreography C, even after its publication.

Similarly, the V&V governance can adopt new test suites also for a specific service on a registry, or for any service enabled to play a given role within a choreography.

Whenever a new test suite is available on a Test Suite Repository (either for a choreography, for a given role in a choreography, or for a specific service), the V&V governance framework would activate a new on-line testing session by executing all the test cases planned into the new test suite.

Rating management architectures (e.g. based either on trust, or reputation models) for decision support and service selection can exploit the on-line validation features offered by the V&V governance framework. As an example, the Governance Framework can agree on the admissible ranking rates that each service participating in a choreography must abide. As V&V strategy, the V&V governance framework could trigger on-line testing sessions whenever the rank of a service reduced below such thresholds. A description of rating metrics adopted within the V&V governance framework follows.

6.2. V&V Rating Policies

Section 6.1 describes how the on-line validation of a service that is indexed by a service registry can be exploited in order to build a trustful environment. This section describes the policies adopted within the

V&V governance framework in order to rate both a single service that is bound to a choreography C, and also C as a whole. Such policies can be partitioned in two main categories : rating policies based on objective testing scores; and rating policies based on subjective evaluation (i.e. feedbacks). The partition depends on the kind of the rating metric each policy refer.

In a scenario that foresees some perpetual validation techniques, the analysis of the results of each testing session gives the possibility to quantify the trustworthiness values according to testing goals. Thus, it allows any requester to the registry to determine which service is more trustable according to an objective estimation (i.e. test passed VS. test failed). Several trust models could be associated to this V&V governance aspect; in Section 6.3.2 we detail a metric based on testing result that we will refer within CHOReOS.

Similarly to the rating of the service trustworthiness on the basis of the results of an on-line testing session, the V&V governance framework could refer to some objective trust models in order to estimate a potential trustworthiness score for the whole choreography. In particular, for each actor A defined in C, the trust model could consider the trustworthiness based on testing score of all the services on the registry that can play as A. The trustworthiness of the whole C is a function of the testing scores computed for all the roles, and it could be interpreted as an objective potential mood (e.g. a benchmark) of the choreography. A concrete metric that instantiates this trust model is described in Section 6.3.1.

The basic idea of a reputation system is to let parties rate each other, for example after the completion of an interaction, and use the aggregated ratings about a given party to derive a reputation score, which can assist other parties in deciding whether or not to interact with that party in the future [38]. Currently, reputation systems represent an interesting and significant trend in decision support, service provision, and service selection (e.g. the Google's +1 Button, the eBay's feedback forum, the Facebook's Thumbs Up button, the LinkedIn's Recommendations).

As argued in [38], in this context we consider service reputation as a metric of how a service is generally perceived its users. Thus, differently from the testing-based trust systems described above that are based on objective measures, here reputation systems are based on subjective judgments.

As described in Section 3.2, the V&V governance framework supports service reputation as a means to determine which service is more reliable on a given registry according to service user's feedback. Several and configurable reputation models could be associated to this V&V governance aspect; in particular Section 6.3.3 describes a model implementing this kind of reputation rule.

Also in this case, the V&V governance framework could refer to some compositional model for user's feedbacks in order to estimate a potential reputation score for the whole choreography. In particular, for each actor A defined in C, the trust model could consider the reputation score (i.e. positive feedbacks vs. negative feedbacks) of all the services on the registry that can play as R. The reputation score of C (as a whole) is a function of the feedback scores computed for all the roles. Thus here, the reputation score of C is interpreted as a benchmark of subjective judgments for the choreography. When the service reputation function described in Section 6.3.3 is referred, the metric described in Section 6.3.1 also implements this choreography reputation model.

6.3. Ranking Rules

This section describes possible ranking strategies that will be coded into ranking rules for the V&V governance policies of CHOReOS.

The next Section 6.3.1 describes a rule for ranking a service choreography according to both the topology of the choreography itself and the ranking of the each available/known service that can play any of the roles specified by the choreography.

The definition of the ranking function applied to the services impacts both on the score and on the interpretation of the ranking function defined for the choreographies. In the following we provide two possible strategies for calculating the service ranking function. Specifically, according to the V&V Rating Policies described in Section 6.2, Section 6.3.2 describes a ranking strategy that is based on the

objective results of the testing sessions; while Section 6.3.3 describes a ranking strategy that based on a subjective reputation model.

6.3.1. Choreography Rank

Differently from the others web search engines that have been used until year 2000, Google dominated the market due to the superior search results that its innovative PageRank algorithm [45] delivered. As described in [38], the PageRank algorithm can be considered as a reputation mechanism because it ranks the web pages according to the number of other pages resulting pointing at it. In other words, the algorithm interprets the collection of hyperlinks to a given page as public information that can be combined to derive an objective reputation score.

In the original definition [45], the PageRank algorithm considers a collection of web pages as a graph where each node is a web page, and the hyperlinks of the pages where modeled as outgoing edges of the graph. In this sense, the strategy proposed by PageRank algorithm can be applied to any problem that can be formulated in terms of a graph.

The ranking metric for a service choreography described in the following is a interpretation of the PageRank algorithm based on both the services involved in a choreography, and on the graph that the choreography subsumes.

Specifically, let us denote S as a set of services. Thus, we define :

$$\mathcal{A}_{\mathsf{C}} = \{A | A \text{ is an actor in } \mathsf{C}\} \tag{6.1}$$

as the set of all the actors defined in C, and:

$$\Omega_{\mathsf{C}}(A) = \{\omega \in \mathsf{S} | A \in \mathcal{A}_{\mathsf{C}}, \omega \text{ plays } A \text{ in } \mathsf{C}\} \tag{6.2}$$

as the set of all the services in S that can play the role A in C. Also, given a relation of dependency among the actors in a choreography, for each actor A in C we denote both the set of actors in C on which A depends (i.e. $N^{+}_{\mathsf{C}}(A)$), and the set of actors in C that depend on A (i.e. $N^{-}_{\mathsf{C}}(A)$). Specifically:

$$N^{+}_{\mathsf{C}}(A) = \{B | A \in \mathcal{A}_{\mathsf{C}}, B \in \mathcal{A}_{\mathsf{C}}, \exists \text{ dependency from } A \text{ to } B \text{ in } \mathsf{C}\} \tag{6.3}$$

$$N^{-}_{\mathsf{C}}(A) = \{B | A \in \mathcal{A}_{\mathsf{C}}, B \in \mathcal{A}_{\mathsf{C}}, \exists \text{ dependency from } B \text{ to } A \text{ in } \mathsf{C}\} \tag{6.4}$$

Section 6.3.4 gives a simple example of the sets $N^{+}_{\mathsf{C}}(A)$, and $N^{-}_{\mathsf{C}}(A)$ on a specific dependency relation. Note that such relation will be actually adopted within the V&V governance framework.

Let us denote R as a ranking function for a given service (see Section 6.3.2, and Section 6.3.3), thus we define the ranking function of an actor A in a choreography C as:

$$\mathcal{R}(A, t, \mathsf{C}) = \begin{cases} \lambda \dfrac{\sum\limits_{\omega \in \Omega_{\mathsf{C}}(A)} R(\omega, t)}{\left|\Omega_{\mathsf{C}}(A)\right|} + \gamma \sum\limits_{B \in N^{+}_{\mathsf{C}}(A)} \dfrac{\mathcal{R}(B, t-1, \mathsf{C})}{\left|N^{-}_{\mathsf{C}}(B)\right|} & \text{for } t \geq 1 \\ \varphi & \text{else} \end{cases} \tag{6.5}$$

Given a role A in choreography C, the case $t = 0$ in Equation 6.5 defines the initial ranking condition for the role A, while the recursive definition for the case $t \geq 1$ is composed by two terms whose interpretation is given below. The parameters λ, and γ can be used in order to tune the contribution of each term in the computation of the actor's ranking function.

The first term of Equation 6.5 gives rank values based on the ranking of the all services that are implementing A in C: the more the services that can play A in C rank well, the better A will perform in

the choreography. The effect of such impact is normalized according to the number of services playing the role A. On the other hand, the first term computes a poor ranking to A if there exist only few and bad ranked services that can play A in C. In other words, this case matches when A could be considered critical for the enactment of C.

The second term in Equation 6.5 gives rank values as a function of the other actors in C on which A depends. In other words, we consider that if the behaviour foreseen for A in C relies on the actions performed by another actor B (e.g. B is the initiator of a task with A), then the rank values scored by B should impact the ranking of A within the whole C. Even though this metric generally helpful, we consider it even more significant when B is bad ranked, for example because of most of the services playing B are not reliable. Note that, as the metric in Equation 6.5 does not take into account the enactment status of the choreography C, the effect of how much B impacts on A is proportionally calculated by considering the number of all the actors that depend form B.

Finally, we denote the ranking function for the whole choraography C as:

$$\mathbb{R}(\mathsf{C}, t) = \sum_{A \in \mathcal{A}_{\mathsf{C}}} \mathcal{R}(A, t, \mathsf{C}) \tag{6.6}$$

6.3.2. Testing-based Service Rank

As introduced in Section 6.2, in the perpetual validation scenario foreseen by the V&V framework, the analysis of the results of each testing session can be used for building a trust model where the trustworthiness of services is ranked in terms values of testing goals.

The very general idea of this assumption is that data from testing results (i.e. both test passed, and test failed) represents quantitative facts that permit to any service in a choreography to determine which other service playing a given role is more trustable according to an objective estimation. For example, if the testing sessions that are executed focus on integration issues, the testing-based service rank explains how a service is behaving with respect to the scenario foreseen by a choreography.

Furthermore, as service behaviour may continuously evolve (e.g. change in the implementation, dynamic binding with other services), a trust model should consider that the closer in time a service is tested the more reliable are the results obtained from the testing session. Thus the definition of a testing-based ranking for services should decrease over time.

The logistic function is a well-studied mathematical function that was originally proposed in [54], and is often used in ecology for modeling population growth. Specifically, in one of its most common formulation, the logistic function offers a non linear evolution of the population function P over the parameter t (e.g. the time) depending on the two parameters: the carrying capacity K, and the growth rate r.

The testing-based ranking we propose defines a function of trust basing on the logistic function, where K is interpreted as the highest admissible level of trust, while for each service w, r is the number of the tests passed over the total number of test executed (see Equation 6.7).

$$r_\omega = \frac{\#\text{passedTest}_\omega}{\#\text{runTest}_\omega} \tag{6.7}$$

For a given service $\omega \in \Omega_{\mathsf{C}}(A)$ that can play the role A in C, Equation 6.8 defines a test-based trust model based on the logistic function.

$$\mathcal{T}(t, \omega) = \frac{K v_{\text{Fade}} e^{(-r_\omega (t - h_{\text{Offset}}))}}{K + v_{\text{Fade}} (e^{(-r_\omega (t - h_{\text{Offset}}))} - 1)} \tag{6.8}$$

Specifically, in Equation 6.8, the h_{Offset}, and the v_{Fade} are configurable parameters useful for translating, and fading the values returned by the trust model. In addition, as the trust model $\mathcal{T}$ is actually an instantiation of the logistic function, the setting of the parameters for $\mathcal{T}$ must keep satisfying the stability criteria foreseen by logistic function (e.g. $K > 1$).

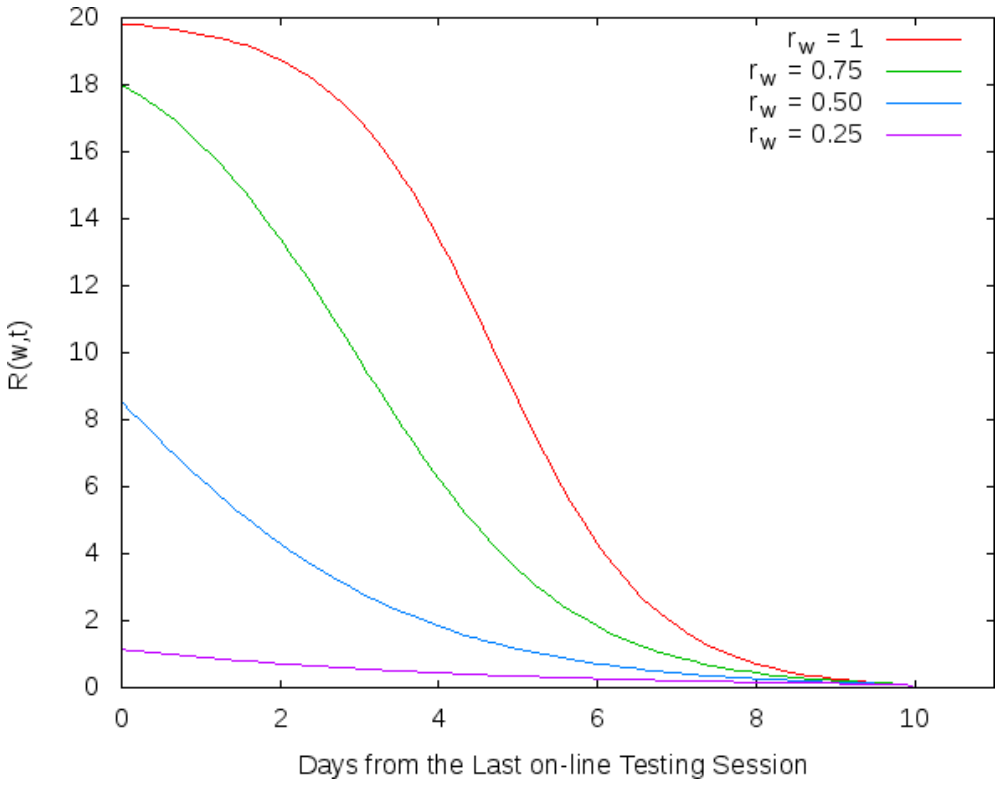

Figure 6.1: Examples of the Evolution of the Testing-based Service Rank Function

Finally, Equation 6.9 defines a testing-based ranking function for a given service ω playing a given role A in C (i.e. $\omega \in \Omega_{\mathsf{C}}(A)$). As introduced in Section 6.3.1, such service-level ranking function can be exploited in order to compute the testing-based rank of the actor A (see Equation 6.5), and consequently the testing-rank of the whole choreography C (see Equation 6.6).

$$R(t,\omega) = \begin{cases} \mathcal{T}(t,\omega) & \text{if } t < h_{\text{Offset}} \\ \\ 0 & \text{else} \end{cases} \tag{6.9}$$

According to the definition in Equation 6.9, the configurable parameter h_{Offset} is interpreted as the maximum observation period (e.g days) after that the testing-based service rank is considered not sufficiently reliable, and than a new testing session for the service is recommended.

As an example, Figure 6.1 depicts the evolution of the function $R(t,\omega)$ with respect to t (i.e. time), and by considering different values of r_ω. Specifically the figure shows the case all the test cases passed (i.e. $r_\omega = 1$), the case some of the test cases passed (i.e. $r_\omega = 0.75$), the case half of the test cases passed (i.e. $r_\omega = 0.50$), and the case most of the test cases failed (i.e. $r_\omega = 0.25$).

6.3.3. Reputation-based Service Rank

The testing-based service rank proposed in Section 6.3.2 offers a quantifiable, and objective metric evaluating how a service is behaving; for example with respect to the scenario envisioned by a service choreography. Differently, reputation systems based on feedbacks provide a widely adopted solution in order to share subjective evaluation of a service after the direct experience of its users.

In the literature several ranking models have been proposed in order to combine user's feedbacks and derive reputation ratings [38]. Among the others, in this section we will refer to a reputation model based on the Beta Density function (β), and originally proposed in [37]. Specifically, the authors in [37] argue how reputation system based on the β function are both flexible, and relatively simple to implement in practical applications. Furthermore, such systems have good foundation on the theory of statistics.

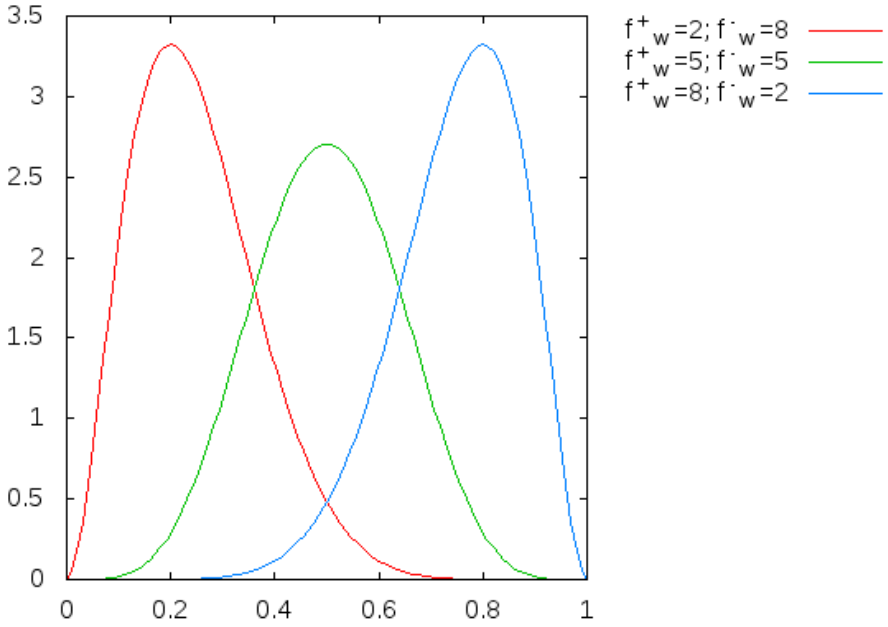

Figure 6.2: Examples of the β Function

Let us consider a service ω playing a given role A in C (i.e. $\omega \in \Omega_{\mathsf{C}}(A)$). Than, we denote $f_\omega^+, f_\omega^- \geq 0$ as the number of positive and negative feedbacks collected by the service ω, respectively.

According with the formulation given in [37], the β function can be written as reported in Equation 6.10, where Γ is the well-studied Gamma function.

$$\beta(p, f_\omega^+, f_\omega^-) = \frac{\Gamma(f_\omega^+ + f_\omega^- + 2)}{\Gamma(f_\omega^+ + 1) * \Gamma(f_\omega^- + 1)} * p^{f_\omega^+} * (1-p)^{f_\omega^-} \tag{6.10}$$

The interesting consideration about the β function is that its mathematical expectation is trivial to compute, and it is given by the Equation 6.11.

$$E(\beta(p, f_\omega^+, f_\omega^-)) = \frac{f_\omega^+ + 1}{f_\omega^+ + f_\omega^- + 2} \tag{6.11}$$

In other words, the feedbacks that the users reported during past interactions with a service ω are interpreted by Equation 6.11 as the probability of collecting a positive feedback with ω on average in the future interactions. For example, if E for the service ω is 0.8 means that is still uncertain if ω will collect a positive feedback in the future (i.e. due to a positive interaction), but it is likely that this would happen.

Figure 6.2 depicts three instantiation of the β function: the case most of the feedback are negatives (i.e. $f_\omega^+ = 2, f_\omega^- = 8$), the case half of the feedback are positives (i.e. $f_\omega^+ = 5, f_\omega^- = 5$), and the case most of the feedback are positives (i.e. $f_\omega^+ = 8, f_\omega^- = 2$).

Finally, Equation 6.12 defines a reputation ranking function based on user's feedbacks for a given service ω playing a given role A in C (i.e. $\omega \in \Omega_{\mathsf{C}}(A)$).

$$R(t, \omega) = E(\beta(p, f_\omega^+, f_\omega^-)) \tag{6.12}$$

As introduced in Section 6.3.1, also such service-level ranking function can be exploited in order to compute the testing-based rank of the actor A (see Equation 6.5), and consequently the testing-rank of the whole choreography C (see Equation 6.6).

Task	$Part$	$Init$
①	*Passenger*, *Airline*	*Airline*
②	*Passenger*, *TravelBookingAgency*	*Passenger*
③	*Passenger*, *TravelBookingAgency*	*TravelBookingAgency*
④	*Passenger*, *TravelBookingAgency*	*Passenger*
⑤	*Passenger*, *TravelBookingAgency*	*Passenger*
⑥	*Passenger*, *TravelBookingAgency*	*TravelBookingAgency*
⑦	*Passenger*, *TravelBookingAgency*	*TravelBookingAgency*
⑧	*Passenger*, *Airline*	*Passenger*

Table 6.1: Instantiation of $Part$, and $Init$

6.3.4. A Simple Example about Rankings

Section 6.3.1 defines the ranking function of an actor in terms of an abstract notion of dependency that can occour among the actors belonging to a choreography. Specifically, for each actor A in a choreography C, such dependency is referred in order to compute both the sets $N^{+}_{\mathsf{C}}(A)$, and $N^{-}_{\mathsf{C}}(A)$. In the following we describe a dependency relation that will be implemented within the V&V governance framework.

Let us consider a choreography C, and let us denote the set of task defined within C:

$$T^{\mathsf{C}} = \{\tau | \tau \text{ is a of tasks defined in C}\} \tag{6.13}$$

the set of actors participating in a given task of C:

$$Part(\tau, \mathsf{C}) = \{A | A \in \mathcal{A}_{\mathsf{C}}, \tau \in T^{\mathsf{C}}, A \text{ is involved in } \tau\} \tag{6.14}$$

and, for each task τ in C, $Init(task)$ is the actor that initiates a task τ.

Thus, given A, B roles in C, the dependency definition we propose relates A with B if and only if B is the initiator of a task were also A is involved. In other words, in order to accomplish a given task in C, A requires some action from B. More formally, we denote this dependency relation $\rightsquigarrow^{C}$ on C as:

$$\rightsquigarrow^{\mathsf{C}} \subseteq \mathcal{A}_{\mathsf{C}} \times \mathcal{A}_{\mathsf{C}} \quad \text{so that} \quad A1 \rightsquigarrow^{\mathsf{C}} A2 \Leftrightarrow \exists \tau \in T^{\mathsf{C}}; A1 \neq A2; A1, A2 \in Part(t, \mathsf{C}); A2 = Init(\tau) \tag{6.15}$$

For example, Figure 6.3 depicts a simple service choreography modeled with the BPMN Choreography Diagram notation. With respect to this example, the sets described above are instantiated as it follows:

- $\mathcal{A}_{\mathsf{C}} = \{$ *Passenger*, *Airline*, *TravelBookingAgency*$\}$
- $T^{\mathsf{C}} = \{$①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧,$\}$

In addition, for each task $\tau \in T^{\mathsf{C}}$, Table 6.1 reports both the set $Part(\tau, \mathsf{C})$, and $Init(\tau)$; while Figure 6.4 depicts the dependency graph resulting from the instantiation of the relation $\rightsquigarrow^{\mathsf{C}}$ on this simple example. Finally, according to the definitions given in both Equation 6.3, and Equation 6.4, Table 6.2 reports the sets $N^{+}_{\mathsf{C}}(A)$, and $N^{-}_{\mathsf{C}}(A)$ resulting form the specific instantiation of the dependency relation here adopted.

6.4. Choreography Enactment Policies

When the design of a choreography is completed, its specification could be indexed by a dedicated registry. As introduced in Section 3.2, the Governance Registry permits to store and retrieve choreography

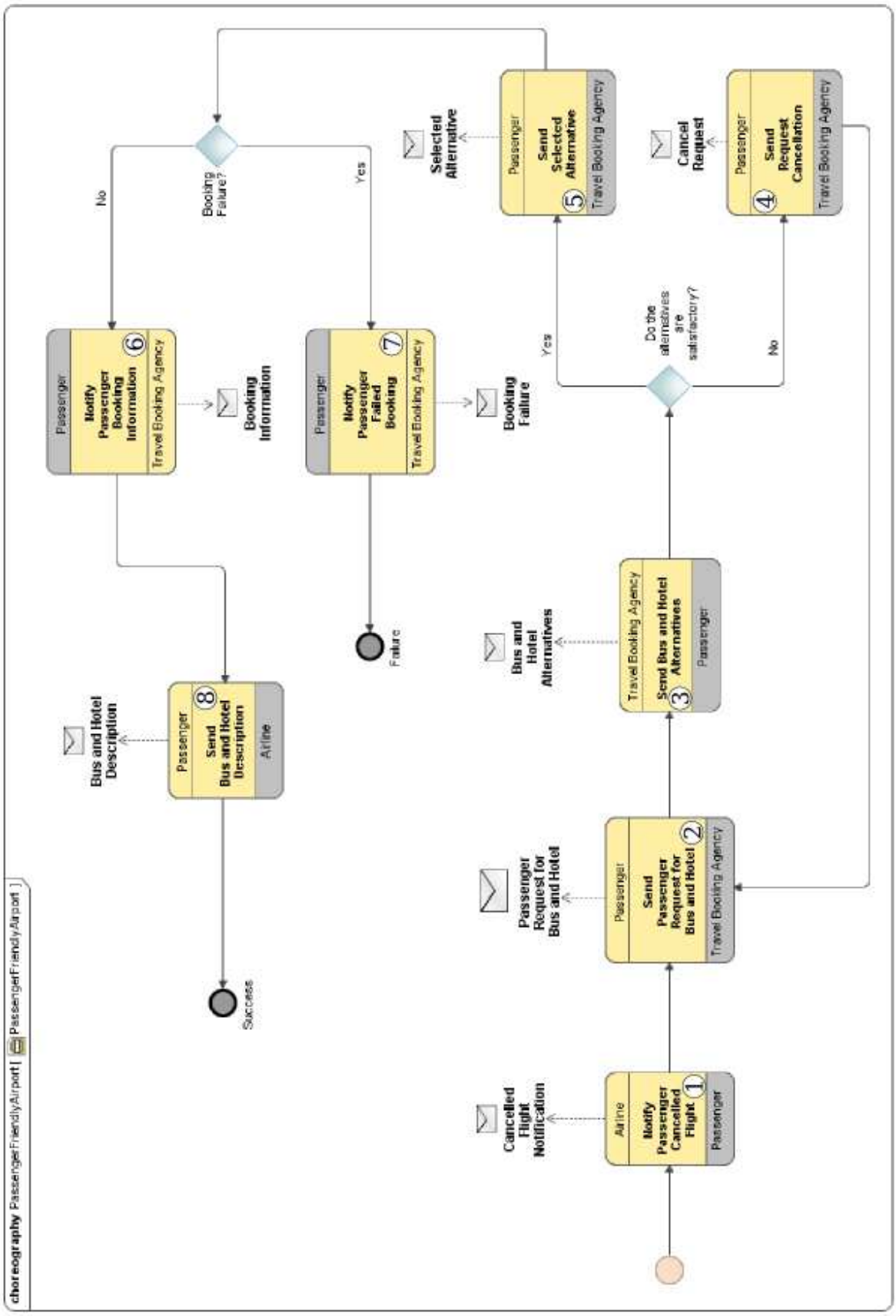

Figure 6.3: Example from the Passenger Friendly Airport Choreography

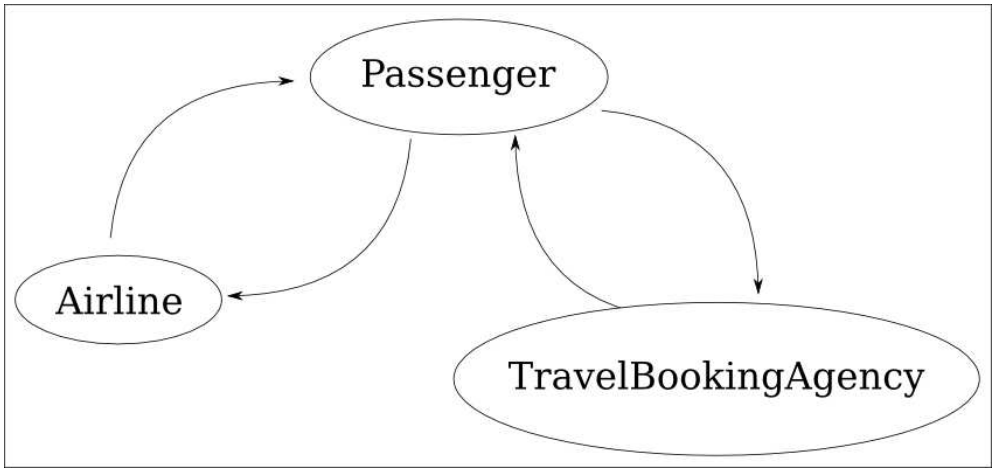

Figure 6.4: Dependency Graph According to the Relation $\rightsquigarrow^{C}$

N^{+}_{C}(*Passenger*)	*Airline*, *TravelBookingAgency*
N^{-}_{C}(*Passenger*)	*Airline*, *TravelBookingAgency*
N^{+}_{C}(*Airline*)	*Passenger*
N^{-}_{C}(*Airline*)	*Passenger*
N^{+}_{C}(*TravelBookingAgency*)	*Passenger*
N^{-}_{C}(*TravelBookingAgency*)	*Passenger*

Table 6.2: Examples of $N^{+}_{C}(A)$, and $N^{-}_{C}(A)$

specifications. Also, the Governance Registry stores information about the status of a choreography. For example, if a given choreography C is enactable or not.

Many kind of strategies could be applied in order to classify a choreography as enactable. For example, a policy defines a choreography C enactable if for any role A in C, the V&V governance framework can point to a set of services (i.e. one or more services) that can play A in C. Section 6.2 describes how the V&V governance framework provides rating mechanisms for both services and choreographies. Thus, enactment policies for C could be also regulated in terms of the rating scores evaluated for the choreography. For example, C is enactable if and only if it scores either a minimal trust (i.e. based on tests), or a minimal reputation (i.e. based on feedbacks) level.

6.5. Test Cases Selection Policies

In literature various policies have been proposed in order to identify a proper test suite for testing a third party service. For example in [52], the authors suggested that for testing a service implementation, integrators should directly access and use test cases provided by the same Service Provider.

In [30] [15], the authors proposed testable services as a solution to provide third-party testers with structural coverage information after a test session, yet without revealing their internal details.

The main drawback of such approaches is that both integrators and service testers do not have enough information to improve their test set when they get a low coverage measure because they do not know which test requirements have not been covered. In [29] the authors propose an approach in which testable services are provided along with test metadata that will help their testers to get a higher coverage. Such idea could fit well the needs of documentation for testing third party services joining a choreography. Therefore, we intend to specify metadata templates which service providers should fill for making their service testable within Choreos.

All the policies described above select test cases from the test suite provided by the Service Provider. In some cases, such approaches could not provide completely objective test suites focusing on integration aspects.

An alternative approach is the definition of test cases selection policies that enable the derivation of test cases from models provided by the Service Provider, for example during the registration of the service. Specifically, within the CHOReOS project, it could be equally useful to derive test cases from the service choreography. In fact, a choreography specification defines both the coordination scenarios in which a service under registration plays a role, and the abstract behaviour expected by each role.

6.6. Ultra-Large Dimension Mitigation Policies for V&V Activities

The discussion about FI challenges reported in Chapter 2, introduces the characterization of ULS dimensions of software systems given in [48]. In the cited report the attribute “ultra-large” is considered as applicable to any aspect of a software system (e.g. size of codes, amount of handled data, number of hardware elements, etc.).

Considering a service choreography C, obviously this might be considered huge in terms of the number of participant actors or of the included activities. However, just taking into account only such two size aspects would be limiting.

Rather, we consider even more interesting the case when C specifies the potential coordination of a huge number of possible services. In other words, we consider a choreography C “ultra large” also when, given any role A in C, it can be huge the number of services that could play that role.

Concerning the first characterization of ultra-large, i.e. the size aspects of a choreography, the impact on governance and V&V is relative, in that the proposed approaches do not change, but we need to evaluate the scalability of the proposed approaches. This will be done as we go ahead with implementation of the approaches, by validating them on some CHOReOS demonstration scenario. We need to ensure that the critical components of the Governance and V&V framework can face the dimensions of the CHOReOS target systems. However, the approaches we propose are on-line and naturally distributed, including distributed registries, distributed monitoring components, and we conceive them already with scalability concerns in mind.

The V&V policies described in the previous sections give broad indications on how the Governance Framework should support V&V activities (e.g. policy for the on-line activation of the V&V sessions, policies for rating the quality of a service choreography, etc.). The application of these policies in principle refer to all the services related to a choreography. However, as a consequence of the growing number of such services, the V&V Manager and the Governance Manager can introduce sampling strategies in order to reduce the application of the Governance Activities at run-time, so to mitigate their costs. Thus, each of the proposed V&V policies, e.g., activation, rating, enactment, etc, could be transformed into a corresponding statistical version, in which the proposed rules are applied according to some random sampling function. In this direction, an interesting approach in the literature is [60], in which the authors propose and make an empirical evaluation of the on-line assessment of reliability of a set of service compositions, by exploiting the data reported from the execution of other similar services. This idea presupposes a collaborative approach to V&V of non-functional properties, which can be put in place only thanks to an established policy framework.

We can also address scalability of QoS attributes in advance, already at design-time, by introducing suitable analysis techniques. In CHOReOS we are building statistically reliable models on the basis of the analysis of adaptable ULS choreographies in the context of WP 2 research. We describe this in Chapter 5 of Deliverable D2.1 [14].

On the other hand the components foreseen by the Governance Framework, and described in Section 3.2, are inherently scalable and would be part of a distributed infrastructure that perpetually enable and regulate the on-line V&V activities. As we will better describe in the next deliverables (e.g. D4.2), such infrastructure would mitigate the effect of the ultra-large dimension by exploiting both grid, and

cloud computing. In this scenario, V&V Manager should design policies regulating the deployment of components (e.g. the Test Driver, the Test Oracle, and the Mocks Factory) over a distributed platform.

In conclusion CHOReOS Governance and V&V supporting mechanisms will be defined considering particular dimensions of ultra large scale with respect to the choreography concepts in a future internet settings. The answers to the posed challenges are mainly based on a highly scalable and distributed architecture and on the support and conceivement of stochastic strategies.

7 Conclusions and Future Work

The FI and its ULS dimensions pose relevant challenges to the definition of a clear and effective governance. The inherent multi-party nature of the FI asks for the introduction of novel mechanisms and tools to enable the definition, monitoring and enforcement of policies and rules. On the other side traditional software engineering activities and in particular V&V related activities need to be rethought in order to face the new challenges and take advantage of possible new opportunities.

This document reported the findings of the research effort carried on by CHOReOS WP 4 on such subjects. With respect to governance, we have first focused on identifying the new roles and responsibilities emerging in this context. In particular we analyzed the implications that the introduction of the choreography concept bring to the management of the ULS dimension. Importantly, the introduction of this new concept asks naturally for the definition of suitable mechanisms to manage its life-cycle, run-time and evolution.

We emphasized the emergence of the need for the introduction of suitable mechanisms to enable V&V activities for FI choreographies, also to permit their extension to the run-time phase. In particular the introduction of V&V activities at run-time requires the definition of precise policies and rules that need to be generally accepted, also in order to manage or avoid possible side effects. At the same time a goal of WP 4 is to explore the emergence of new roles in V&V activities, to be played when extended to the run-time phase. Considering the very nature of FI as a socio-technical system, WP 4 envisages the possibility of using V&V techniques in order to augment trustworthiness on artifacts related qualities by their potential users [19].

This document reported the results of the investigations carried on within WP 4 in the first months of its activity. Its objective is mainly to identify the basic concepts and a research route for the future activities to be carried on by the workpackage. Starting from this document the partners started to implement the envisaged mechanisms and requirements and a first set of components to implement has been already identified and introduced in this document. Several presented results provide promising feature to discipline and validate choreography-centric development. Among others, we proposed a preliminary architecture of the governance and V&V framework. We introduced an original classification of V&V policies, and outlined an approach to model and analyse SLA-related policies. We started to compile a list of policies for testing and ranking of services playing a role in a specified choreography.

However, as we said in the introduction, this document should be received as a draft first set of policies to be iterated upon on a continuous basis in cooperation with the whole CHOReOS consortium. The policies that we propose constitute a principled list of required good practices, and are purposely not yet finalized into a formal description. In fact, governance is a transversal concept behind the CHOReOS IDRE under development, and needs to be harmonized with the processes and tools developed in the other WPs.

Thus, the release of such document will be followed by a comparison and coordination with the other results simultaneously delivered. The revised set of policies and the supporting mechanisms will be then incorporated into the next WP 4 deliverables. In particular, the next-coming Deliverable D4.2 (due at Month 18) will provide a first set of implementations for the specified mechanisms and policies.

Bibliography

[1] Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0/.

[2] SOAP: Simple Object Access Protocol. http://www.w3.org/TR/soap/.

[3] USDL: Unified Service Description Language. http://www.internet-of-services.com/index.php?id=288&L=0.

[4] WS-Agreement: Web Service Agreement. www.ogf.org/documents/GFD.107.pdf.

[5] WS-BPEL: Web Service Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[6] WS-I: Web Service Interoperability. http://www.ws-i.org/.

[7] WS-Policy: Web Service Policy – Framework. http://www.w3.org/TR/ws-policy.

[8] WS-Policy: Web Service Policy – Primer. http://www.w3.org/TR/ws-policy-primer/.

[9] WSDL: Web Service Description Language. http://www.w3.org/TR/wsdl.

[10] XACML: eXtensible Access Control Markup Language. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[11] A. Anderson and S. Proctor. XACML profile for Web-services. *Structure*, 16:52, 2003.

[12] A.H. Anderson. An introduction to the web services policy language (wspl). In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 189–192. IEEE, 2004.

[13] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement). In *Global Grid Forum*. The Global Grid Forum (GGF), 2004.

[14] M. Autili and M. Tivoli, editors. *D2.1 : CHOReOS dynamic development model definition*. The CHOReOS Consortium, Oct. 2011.

[15] C. Bartolini, A. Bertolino, S.G. Elbaum, and E. Marchetti. Bringing white-box testing to service oriented architectures through a service oriented approach. *Journal of Systems and Software*, 84(4):655–668, 2011.

[16] K. Beck. *Test-driven development: by example*. Addison-Wesley, Boston, 2003.

[17] A. Ben Hamida, editor. *D5.2 : Specification of the CHOReOS IDRE*. The CHOReOS Consortium, Oct. 2011.

[18] J. Bernhardt and D. Seese. Service-oriented computing — icsoc 2008 workshops. chapter A Conceptual Framework for the Governance of Service-Oriented Architectures, pages 327–338. Springer-Verlag, Berlin, Heidelberg, 2009.

[19] A. Bertolino, G. De Angelis, S. Kellomäki, and A. Polini. Enhancing Trustworthiness within Service Federations by Continuous On-line Testing. *IEEE Computer*, 2011. – to appear, DOI:10.1109/MC.2011.227.

[20] A. Bertolino, G. De Angelis, and A. Polini. (role)cast : A framework for on-line service testing. In *Proc. of the 7th International Conference on Web Information Systems and Technologies (WEBIST 2011)*, Noordwijkerhout, The Netherlands, May 2011.

[21] A. Bertolino, L. Frantzen, A. Polini, and J. Tretmans. Audition of Web Services for Testing Conformance to Open Specified Protocols. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, number 3938 in Lecture Notes in Computer Science, pages 1–25. Springer, 2006.

[22] A. Bertolino and A. Polini. The audition framework for testing web services interoperability. In *EUROMICRO-SEAA*, pages 134–142. IEEE Computer Society, 2005.

[23] A. Bertolino and A. Polini. Soa test governance: Enabling service integration testing across organization and technology borders. In *Proc. of Software Testing, Verification and Validation Workshops, 2009. ICSTW '09*, pages 277–286, Apr. 2009.

[24] M.B. Blake and D.J. Cummings. Workflow composition of service level agreements. In *Services Computing, 2007. SCC 2007. IEEE International Conference on*, 2007.

[25] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *ESEC/SIGSOFT FSE*, pages 131–140, 2009.

[26] P. Chatel, editor. *D5.1 – Requirements for the CHOReOS IDRE*. The CHOReOS Consortium, Mar. 2011.

[27] A. Di Marco, C. Pompilio, A. Bertolino, A. Calabró, F. Lonetti, and A. Sabetta. Yet another metamodel to specify non-functional properties. In *Int. ECOWS Workshop on Quality Assurance for Service-based applications QASBA 2011*. ACM, Sept. 2011.

[28] eBay Open Source. Turmeric repository. https://www.ebayopensource.org/wiki/display/TURMERICDOC/Repository.

[29] M.M. Eler, A. Bertolino, and P. Masiero. More testable service compositions by test metadata. In *6th IEEE International Symposium on Service-Oriented System Engingeering SOSE 2011*, Washington, DC, USA, Dec. 2011. IEEE Computer Society.

[30] M.M. Eler, M.E. Delamaro, J.C. Maldonado, and P.C. Masiero. Built-in structural testing of web services. In *Proc. of Brazilian Symp. on Soft. Engineering*, pages 70–79, 2010.

[31] T. Erl, R. Laird, R. Schneider, L. Shuster, A. Manes, S.G. Bennett, C. Gee, R. Moores, C. Venable, A. Tost, et al. *Soa Governance: Governing Shared Services On-Premise and in the Cloud*. 2011.

[32] M. Fiedler, A. Gavras, N. Papanikolaou, H. Schaffers, and N. Wainwright. Future Internet Assembly Research Roadmap – Towards Framework 8: Research Priorities for the Future Internet. Technical report, Future Internet Assembly Working Group, May 2011.

[33] V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *J. Syst. Softw.*, 80:528–558, April 2007.

[34] I. Haq, A. Huqqani, and E. Schikuta. Aggregating hierarchical service level agreements in business value networks. In *Proceedings of the 7th International Conference on Business Process Management*, BPM '09, pages 176–192, 2009.

[35] V. Issarny, editor. *D1.1 : CHOReOS State of the Art, Baseline, and Beyond*. The CHOReOS Consortium, Dec. 2010.

[36] V. Issarny, editor. *D1.3 : Initial Architectural Style for CHOReOS Choreographies*. The CHOReOS Consortium, Oct. 2011.

[37] A. Jøsang and R. Ismail. The Beta Reputation System. In *Proc. of the 15th Bled Electronic Commerce Conference*, Jun. 2002.

[38] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[39] J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, 1987.

[40] C.M. MacKenzie, K. Laskey, F. McCabe, P.F. Brown, and R. Metz. Reference model for service-oriented architecture, version 1.0. Technical report, OASIS, 2006.

[41] P. Malinverno. Gartner research index on SOA governance, 2006.

[42] A.T. Manes. Understanding SOA governance, SOA Magazine, Issue XL. `http://www.soamag.com/I40/0610-2.php`, June 2010.

[43] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. *Network and distributed systems management*, pages 303–347, 1994.

[44] M.J. Maullo and S.B. Calo. Policy management: an architecture and approach. In *Systems Management, 1993., Proceedings of the IEEE First International Workshop on*, pages 13 –26, apr 1993.

[45] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report SIDL-WP-1999-0120, Stanford University, November 1999.

[46] D.B. Petriu and C.M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from uml designs. *Software and System Modeling*, 6(2):163–184, 2007.

[47] T. Phan, J. Han, J-G. Schneider, T. Ebringer, and T. Rogers. A survey of policy-based management approaches for service oriented systems. *Software Engineering Conference, Australian*, 0:392–401, 2008.

[48] B. Pollak, editor. *Ultra-Large-Scale Systems – The Software Challenge of the Future*. Software Engineering Institute – Carnegie Mellon, June 2006.

[49] S. Simanta, E. Morris, G.A. Lewis, and D.B. Smith. A framework for assurance in service-oriented environments. In *Systems Conference, 2010 4th Annual IEEE*, pages 547 –552, april 2010.

[50] The OMG. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, June 2011. Doc. Number: formal/2011-06-02.

[51] K.S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*. John Wiley and Sons Ltd., Chichester, UK, 2nd edition edition, 2002.

[52] W.T. Tsai et al. Scenario-based web service testing with distributed agents. *IEICE Transaction on Information and System*, E86-D(10):2130–2144, 2003.

[53] A. Vedamuthu, D. Orchard, M. Hondo, T. Boubez, and P. Yendluri. Web Services Policy 1.5 – Primer. *W3C*, Jun 2007.

[54] P.F. Verhulst. Notice sur la loi que la population poursuit dans son accroissement. *Correspondance mathématique et physique*, 10, 1838.

[55] M. Wheaton. Decorating your soa services with governance enforcement contracts. *Agenda*, 2007.

[56] R. Wies. Using a classification of management policies for policy specification and policy transformation. In *Proceedings of the IFIP/IEEE International Symposium on Integrated network Management*, pages 44–56. Chapman&Hall, 1995.

[57] E. Wustenhoff. Service Level Agreement in the Data Center. `http://www.sun.com/blueprints`, 2002.

[58] Ü. Yalçinalp et al. Web Services Policy 1.5 – Guidelines for Policy Assertion Authors. *W3C Working Group Note*, 12, 2007.

[59] A. Zarras, editor. *D3.1 : CHOReOS Middleware Specification*. The CHOReOS Consortium, Oct. 2011.

[60] Z. Zheng and M.R. Lyu. Collaborative reliability prediction of service-oriented systems. *Proceedings of the 32nd ACMIEEE International Conference on Software Engineering ICSE 10*, 1(ICSE 2010):35, 2010.

[61] X. Zhou, W. T. Tsai, X. Wei, Y. Chen, and B. Xiao. Pi4soa: A policy infrastructure for verification and control of service collaboration. In *Proceedings of the IEEE International Conference on e-Business Engineering*, ICEBE '06, pages 307–314, Washington, DC, USA, 2006. IEEE Computer Society.

[62] Y. C. Zhou, X. Peng Liu, E. Kahan, X. Ning Wang, L. Xue, and K. Xin Zhou. Context aware service policy orchestration. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 936 –943, july 2007.