Deliverable D1.4 (a): Final CHOReOS Conceptual Model and its Relation with the CHOReOS Development Process and IDRE

Marco Autili

ICT IP Project

Deliverable D1.4 (a)

Final CHOReOS Conceptual Model and its Relation with the CHOReOS Development Process and IDRE

http://www.choreos.eu

LaTeX template v. 1.16

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

Deliverable Number	:	D1.4 (a)
Title of Deliverable	:	Final CHOReOS Conceptual Model and its Relation with the CHOReOS Development Process and IDRE
Nature of Deliverable	:	Report
Dissemination level	:	Public
Licence	:	Creative Commons Attribution 3.0 License
Version	:	3.0
Contractual Delivery Date	:	31 March 2013 (Revised)
Actual Delivery Date	:	25 April 2013
Contributing WP	:	WP1
Editor(s)	:	Marco Autili (UDA), Davide Di Ruscio (UDA)
Author(s)	:	James Lockerbie (CITY), Neil Maiden (CITY), Guglielmo De Angelis (CNR), Georgios Bouloukakis (Inria), Nikolaos Georgantas (Inria), Sara Hachem (Inria), Valérie Issarny (Inria), Amira Ben Hamida (Linagora), Davide Di Ruscio (UDA), Marco Autili (UDA), Paola Inverardi (UDA), Andrea Polini (Unicam) Alfredo Goldman (USP), Carlos Eduardo Moreira dos Santos (USP), Daniel de Angelis Cordeiro (USP), Gustavo Ansaldi Oliva (USP), Leonardo Alexandre Ferreira Leite (USP), Apostolos Zarras (UOI),
Reviewer(s)	:	Gustavo Ansaldi Oliva (USP), Sébastien Keller (THALES)

Abstract

This is Part (a) of Deliverable D1.4, which specifies the final CHOReOS conceptual model, that is, a high level common ground capturing the relevant concepts and the relationships, underlying the CHOReOS development process and the supporting infrastructures of choreography-based service-oriented systems of the Future Internet. The conceptual model has been defined by means of an iterative process that started with the representation of preliminarily identified concepts, which were the focus of Deliverable D1.2 [15]. Then, the model has been incrementally extended/refined by using the MOTHIA tool that reduced the knowledge gap between the domain and modeling experts while iterating the refinement process. The CHOReOS conceptual model extends NEXOF-RA by introducing new concepts and refining existing ones. It is organized into nine concerns that will be individually described in the document: *Composition*, *Discovery*, *Analysis*, *Messaging*, *Services*, *Resource*, *Management*, *Presentation*, and *Security*.

Keyword List

Conceptual Modeling, Architectural Style, Coordination, Choreography, Future Internet, Interaction Paradigms, Model Transformation, Model Driven Engineering, Scalability, Interoperability

Document History

Version	Changes	Author(s)
0.1	Outline Draft	Marco Autili (UDA), Davide Di Ruscio (UDA)
1.x	First version and revision of individual chapters	All authors
2.0	Overall integration and edition	Davide Di Ruscio (UDA)
3.0	Final revision	Marco Autili (UDA), Davide Di Ruscio (UDA)

Document Reviews

Review	Date	Ver.	Reviewers	Comments
Outline	21 January 2013	1.0	All authors	Outline agreed by all
Draft	14 March 2013	1.x	Davide Di Ruscio	Intermediate version released for final review
QA	6 April 2013	2.0	Gustavo Ansaldi Oliva (USP), Sébastien Keller (THALES)	Editorial comments
PTC	25 April 2013	A	PTC	-

Glossary, acronyms & abbreviations

Item	Description
BPEL	Business Process Execution Language
BPEL4SWS	Business Process Execution Language for Semantic Web Services
BPMN	Business Process Modeling Notation
BPMN2	Business Process Modeling Notation - ver. 2
CBA	Choreography Based Architecture
CCS	Calculus of Communicating Systems
CS	Client Server
CSP	Communicating Sequential Processes
DE	Domain Expert
DAML	DARPA Agent Markup Language
DAML-S	DARPA Agent Markup Language for Services
DOW	Description of Work
ESB	Enterprise Service Bus
FI	Future Internet
GA	Generic Application
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
IDRE	Integrated Development and Runtime Environment
IOPE	Inputs, Outputs, Preconditions, Effects
JBI	Java Business Integration
JSON	JavaScript Object Notation
LTS	Labeled Transition System
MDE	Model Driven Engineering
ME	Modeling Expert
MOTHIA	Model Testing by Human Interrogations & Answers
NEXOF	NESSI Open Framework
NEXOF-RA	NEXOF Reference Architecture
NL	Natural Language
OASIS	Organization for the Advancement of Structured Information Standards
OWL	Ontology Web Language
OWL-S	Ontology Web Language for Services
PaaS	Platform as a Service
PS	Publish Subscribe
REST	REpresentational State Transfer
S & A	Sensor and Actuators
SaaS	Software as a Service
SAWSDL	Semantically Annotated Web Service Description Language
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

SOM	Service Oriented Middleware
TS	Tuple Space
ULS	Ultra-Large-Scale
ULS-FI	Ultra-Large-Scale Future Internet
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Applications Description Language
WP	Work Package
WSCL	Web Service Conversation Language
WSDL	Web Service Definition Language
WSDL-S	Web Service Description Language with Semantics
WSN	Wireless Sensor Network
WSAN	Wireless Sensor and Actuator Network
WSCL	Web Service Conversation Language
WSQM	Web Service Quality Model
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language

Table Of Contents

List Of Tables

List Of Figures

1 Introduction

This document specifies the final CHOReOS conceptual model, that is, a high level common ground capturing the relevant concepts and the relationships, underlying the CHOReOS development process and the supporting infrastructures of choreography-based service-oriented systems of the Future Internet. Indeed, this is a companion document of Deliverable D1.4 Part (b) [18], which specifies the final CHOReOS architectural style. As detailed in [18], the CHOReOS architectural style specifies the types of components, connector, and configurations that support the development of choreographies for the Future Internet, as enabled by the CHOReOS technologies developed in WP2 to WP4 and integrated in the WP5 IDRE. To this end, the definition of the CHOReOS conceptual model in WP1 has been guided by the challenges posed by the Future Internet (identified in [15, 7]), whose objectives have been addressed in the companion Deliverable D1.4(b), while defining the CHOReOS architectural style, and in the latest deliverables of WP2 to WP5, while defining the CHOReOS development process, the CHOReOS middleware, the V&V and governance framework, as well as, the CHOReOS IDRE.

Since its preliminary version, the CHOReOS conceptual model was born as an extension of the NEXOF Reference Architecture (NEXOF-RA)[1]. The latter defines a pattern-based reference architecture for SOA infrastructures within the Future Internet and, as part of it, defines a conceptual model specifying the first-level entities that constitute these infrastructures, the facilities they provide, as well as relationships among them. However, as detailed in Deliverable D1.2 [15], NEXOF-RA does not emphasize neither the concept of choreography, nor the concepts underlying the development process of choreography-based service-oriented systems and the supporting infrastructure. In fact, within the NEXOF reference architecture, the concepts of choreography is simply considered as a specialization of service composition, without providing a clear definition of it. Instead, CHOReOS aims at investigating the impact of the Future Internet's ULS on the development of choreographies of services and things, and as such introduces new concepts in the NEXOF-RA conceptual model and refines existing ones. Comparing the NEXOF work with the CHOReOS work (see Deliverable D1.2 [15] for details), it can be said that the NEXOF conceptual model and the NEXOF reference architecture concern general service-oriented systems; whereas, the CHOReOS conceptual model and the CHOReOS IDRE architecture concern the class of service-oriented systems realized as choreographies and "architected" according to the CHOReOS architectural style. Moreover, the NEXOF reference architecture is an abstract specification that can be used as a reference for architecting specific and concrete SOA infrastructures; whereas, the CHOReOS IDRE architecture is per se a concrete specification of the SOA infrastructure offered by CHOReOS to support the design, development, enactment, analysis, and validation of choreography-based service-oriented large-scale systems in the Future Internet, i.e., "CHOReOS choreographies".

The NEXOF-RA conceptual model is organized into nine concerns, namely, *Composition*, *Discovery*, *Analysis*, *Messaging*, *Services*, *Resource*, *Management*, *Presentation*, and *Security*. In this document, we describe how the CHOReOS conceptual model extends these concerns and how its final version was defined by following an iterative process that started with the representation of the preliminary concepts identified in Deliverable D1.2 [15]. More precisely, the model was incrementally extended/refined by using the MOTHIA tool presented in [1] that reduced the knowledge gap between the domain and modeling experts while iterating the refinement process.

[1] http://www.nexof-ra.eu/

The content of this document can be summarized as follow:

- Chapter 2 describes the methodology and the supporting tools we adopted to refine the preliminary version of the CHOReOS conceptual model presented in Deliverable D1.2 [15]. In particular, we have extended the approach in [1] and the related supporting tool, which have been originally conceived to validate only Class Diagrams. Such extension allowed us to validate also Use Case and Activity Diagrams, hence enabling a comprehensive validation of all the types of diagrams used to specify the CHOReOS conceptual model. Adopting the extended approach, and being supported by the related tools, we were able to automatically derive, from the diagrams of the conceptual models, a set of customizable questionnaires expressed in Natural Language. Using such questionnaires, we (the Modeling Experts - MEs) interviewed the CHOReOS Domain Experts (DEs) and identified portions of the model that were subject to further adjustments or enhancements. Indeed, an iterative process was adopted to refine the conceptual model by incrementally considering the feedback coming from the CHOReOS DEs. We have exploited a formalization of the semantics of those UML modeling elements constituting the diagrams we used to define the conceptual model, i.e., Class, Use Case, and Activity Diagrams. Thus, for each generated question, it was possible to predict its expected answer by applying semantics inference on the current version of the model. To this end, following the approach in [1], we used a syntactical combination of elements considered relevant in the conceptual model and defined the properties to be used by the supporting tool to explore and query the diagrams of the conceptual model.
- Chapter 3 reports the final conceptual model for CHOReOS as outcome of the refinement process described in Chapter 2. The final conceptual model builds upon the Future Internet characterization given in Deliverable D1.2 [15] and previously published by the CHOReOS consortium in [7], and the baseline NEXOF reference architecture surveyed in [15]. Since its inception, the conceptual model has been providing an integrated view and a common understanding among the different CHOReOS partners, and has served as a high-level input domain model to the development of the RTD work packages WP2 to WP5. At the same time, the model has been refined/extended according to the outcomes of RTD work packages WP2 to WP5, as recalled hereafter:
  - WP2: CHOReOS development process and supporting toolset;
  - WP3: CHOReOS middleware providing the necessary runtime support for choreographies in the ULS Future Internet of Services and Things;
  - WP4: CHOReOS solutions supporting Governance and Verification & Validation of ULS choreographies;
  - WP5: infrastructure architecture of the CHOReOS IDRE, which integrates the results of WP2 to WP4, thereby supporting the design, development, enactment, analysis, and validation of choreography-based large scale service-oriented systems in the Future Internet.
- Chapter 4 concludes the deliverable, with a summary of its contribution.

2 Refinement of the CHOReOS Conceptual Model

The CHOReOS conceptual model was conceived to capture the relevant entities/concepts, and relationships among them, underlying systems realized as choreographies of services and things. As already introduced, the developed domain model is an extension of NEXOF-RA and is organized into different concerns as shown in Fig. 2.1. With reference to Deliverable D1.2 [15], each concern is described by considering three different views: a functional view given as UML Use Case diagrams, a structural view modeled by means of UML Class diagrams, and a data flow view specified by means of UML Activity diagrams.

In this chapter, we describe the methodology (Section 2.1) and the supporting tools (Section 2.2) we have conceived and adopted to refine the preliminary CHOReOS conceptual model presented in Deliverable D1.2 [15]. Section 2.3 and Section 2.4 discuss the feedback obtained by domain experts about the initial version of the model, and how it has been taken into account to produce the new version of the model, respectively.

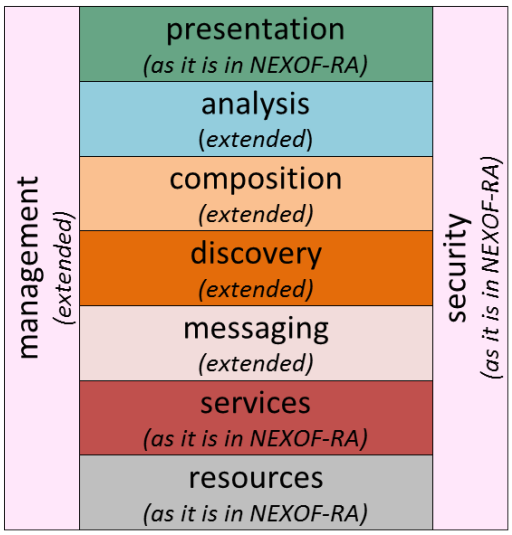


Figure 2.1: Overview of the CHOReOS conceptual model as in D1.2

2.1. Overview of the Methodology Used to Refine the CHOReOS Conceptual Model

Since its initial version, the CHOReOS conceptual model has been conceived by Modeling Experts (MEs) through the interaction with Domain Experts (DEs) to represent the domain of interests in terms of diagrams. The refinement presented in this document has been done mainly to add unforeseen aspects and to correct errors due to the knowledge gap between the DEs and the modeling notation adopted by the MEs. In fact, since the modeling artifacts produced by MEs may be incomprehensible for the DEs, it is usually hard for the DEs to properly validate a candidate solution proposed by MEs.

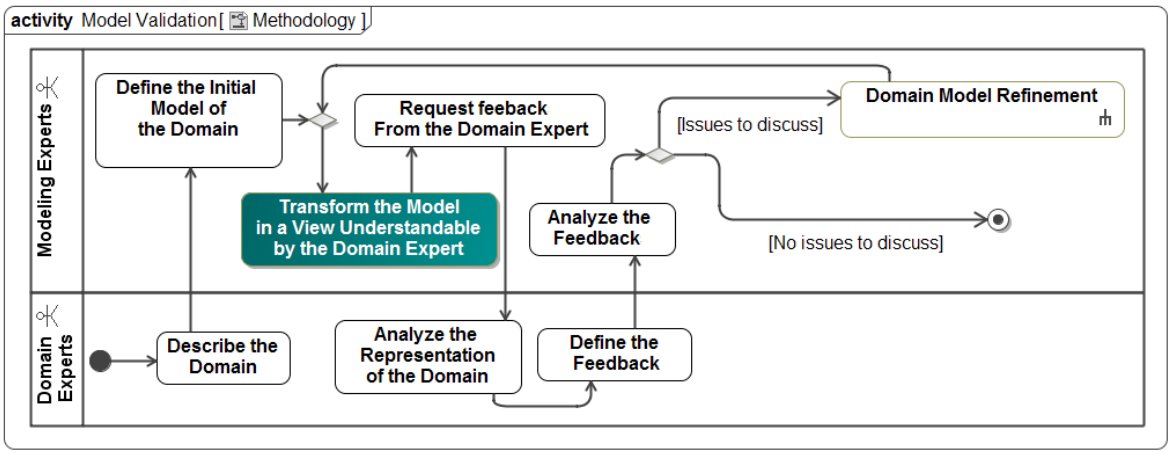


Figure 2.2: Overview of a generic domain modeling process

The refinement of the CHOReOS conceptual model has been performed by following the methodology shown in Fig. 2.2 as a UML Activity Diagram. It models the steps involved in a domain modeling process, with an emphasis on the communications occurring between DEs and MEs.

A common practice in the definition of a domain model is to start with the establishment of a shared vocabulary and the elicitation of a common high-level view of the input domain (see the activity *Describe the Domain* in Fig. 2.2). Usually, after this phase, the MEs can propose an initial model of the domain (see *Define the Initial Model of the Domain*), which is supposed to be validated by the DEs. To this purpose, an iterative process starts. Along these iterations the MEs try to make the model comprehensible to DEs by generating and explaining appropriate model views. Based on such views, MEs request, collect, and analyze the feedback from the DEs to refine the initial model.

Although it is difficult to imagine that these steps can be fully automated, MDE supports their implementation in order to achieve more systematic and more effective results. In particular, with reference to Fig. 2.2, we support the automatic transformation of the model into views comprehensible by DEs (the dark-shaded activity in the figure), by automatically generating a tunable list of simple `Yes/No` questions that span over all model elements.

The idea is that at each iteration of the model validation process, MEs can interview the DEs by means of such automatically generated questionnaires, thus MEs are freed from the difficult task of translating the model into Natural Language (NL) descriptions. Furthermore, such translations is also an error-prone task due to the lack of formal and systematic interpretation of the semantic of the modeling language that has been adopted. For each automatically generated question, it is possible to predict its expected answer by applying *semantics inference* on the current version of the domain model. A detailed description on the generation process of both the questions and their expected answers is reported in Section 2.2.

The subsequent steps concern the feedback analysis and their impact on the domain model. Specifically, if the feedback reveals some mismatches between the domain model and the DEs interpretation (see the gateway after the activity *Analyze The Feedback*), the MEs should understand the nature of the conflict and address it. Using our tool, the MEs can effectively focus on the costly task of discussing feedback on those parts of the model which a conflict belongs to, and propose a new version of the domain model. This process can be iterated until all stakeholders are satisfied with the domain model. The resolution of the identified conflicts is performed in the *Domain Model Refinement* activity consisting of a number of steps as shown in Fig. 2.5 and detailed in Section 2.4.

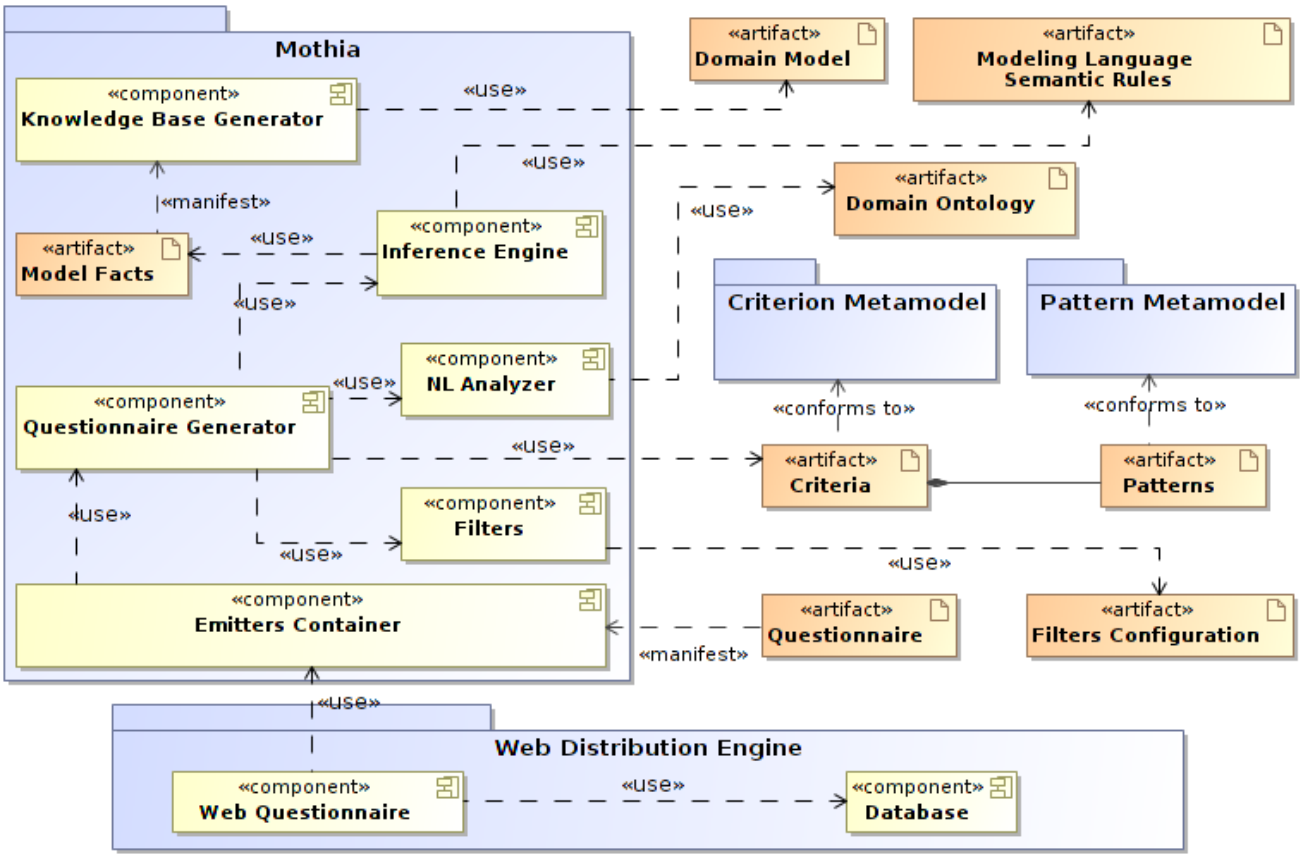


Figure 2.3: The architecture of MOTHIA

2.2. Model Testing by Human Interrogations & Answers

The refinement methodology shown in Fig. 2.2 has been supported by the MOTHIA framework [1]. In this section, we outline the framework's architecture (Section 2.2.1), reference implementation (Section 2.2.2), and Web-based distribution engine (Section 2.2.3), which was employed to collect feedback about the conceptual model from DEs.

2.2.1. The Architecture of MOTHIA

MOTHIA deals with the knowledge gap that separates MEs and DEs: it takes a domain model as input and produces a natural language (NL) questionnaire as output. MOTHIA works on a first-order logic representation of the input model, using the `InferenceEngine` component to check for the satisfiability of desired properties. Fig. 2.3 depicts the internal structure of MOTHIA (left-hand side), and how it interacts with the model (right-hand side) and with the Web Distribution Engine (bottom side).

The `KnowledgeBaseGenerator` is responsible for converting the input domain model into an internal representation describing the domain entities and their relations as facts. The `InferenceEngine` loads such facts and the set of rules representing the semantics of the input modeling language. By querying the `InferenceEngine`, MOTHIA checks whether a given property is valid in the input domain model. Also, the `InferenceEngine` can return all the entities in the domain model satisfying a given property.

The `QuestionnaireGenerator` loads the configurations that drive the queries to the `InferenceEngine`, and the creation of the questionnaire. The configurations of the `QuestionnaireGenerator` are expressed in term of *patterns* and *criteria*:

- The patterns represent syntactical combinations of elements in the input domain model that are considered relevant, i.e., a portion of the specific input domain model.
- The criteria define the properties that MOTHIA uses to explore and query the input domain model. In other words, a criterion abstracts a type of question in the questionnaire and implements it by composing patterns.

The `NL Analyzer` helps with the creation of NL questions by querying a domain *ontology*. It can either enrich and refine the text of a question, or try to infer entities and relationships that are not in the input model. Generated questions fall into three categories: a *Deductive* question is inferred from `true` predicates on the input domain model, a *Distractor* from `false` predicates, and a *Hypothesis* from predicates obtained from the domain ontology.

Querying the input domain model from the set of criteria usually leads to a questionnaire with an intractable number of questions. To reduce the number of questions, the `QuestionnaireGenerator` uses the `Filters` component. A filtering policy can be a random selection of questions, a word-based selection, or a combination of the two (other filterings can be devised).

Finally, the `EmittersContainer` component deals with the output format of the questionnaire. In particular, it can be configured to interact with a `WebDistributionEngine`, in order to publish questionnaires on a web page and collect answers in a `Database`. This is especially useful in geographically distributed projects to ease and enhance cooperation among remote partners.

2.2.2. The Reference Implementation

MOTHIA is an Eclipse-based plugin, written using Java and EMF technologies [2]. MOTHIA can generate questions in NL from a variety of models: ECore diagrams [2], UML Class, Use Case, and Activity diagrams.

According to the abstract architecture described in Section 2.2.1, the reference implementation uses SWI-Prolog as the `InferenceEngine` and WordNet [11] as the lexical ontology for the `NL Analyzer`. The `KnowledgeBaseGenerator` currently handles a selection of the most relevant types of model elements and converts them to Prolog facts.

In addition, the `KnowledgeBaseGenerator` can also codify sets of Prolog rules implementing the semantics of the supported input languages (e.g., the Liskov substitution principle [8] for generalizations in both Class and Use-Case diagrams). Any arbitrary or domain specific semantic interpretation of the model can be easily implemented and plugged in this component.

The facts in the knowledge base are the building blocks to create patterns. As previously stated, a pattern is a syntactical structure in the input model. Facts are therefore composed together to form a pattern by means of a Prolog query. A pattern constructed this way can itself be used to compose subsequent patterns, allowing for arbitrarily complex yet manageable structures.

After patterns are identified in the model, criteria can use them to construct a set of NL questions. In our reference implementation, we defined criteria based on both the literature on model validation and on our own experience in modeling.

2.2.3. A Web-based Engine for Consuming the MOTHIA's Artifacts

Another feature offered by MOTHIA is abQuestionnaire, a web-based engine supporting the distribution of questionnaires and the collection of their feedback. In fact, this engine is the reference implementation of the abstract component: `WebDistributionEngine`, presented in Section 2.2.1. In practice, this component become an important asset for our approach, since people involved in the modeling activity belonged to different organizations and limited face-to-face interactions could be organized (this was the case in the context of CHOReOS).

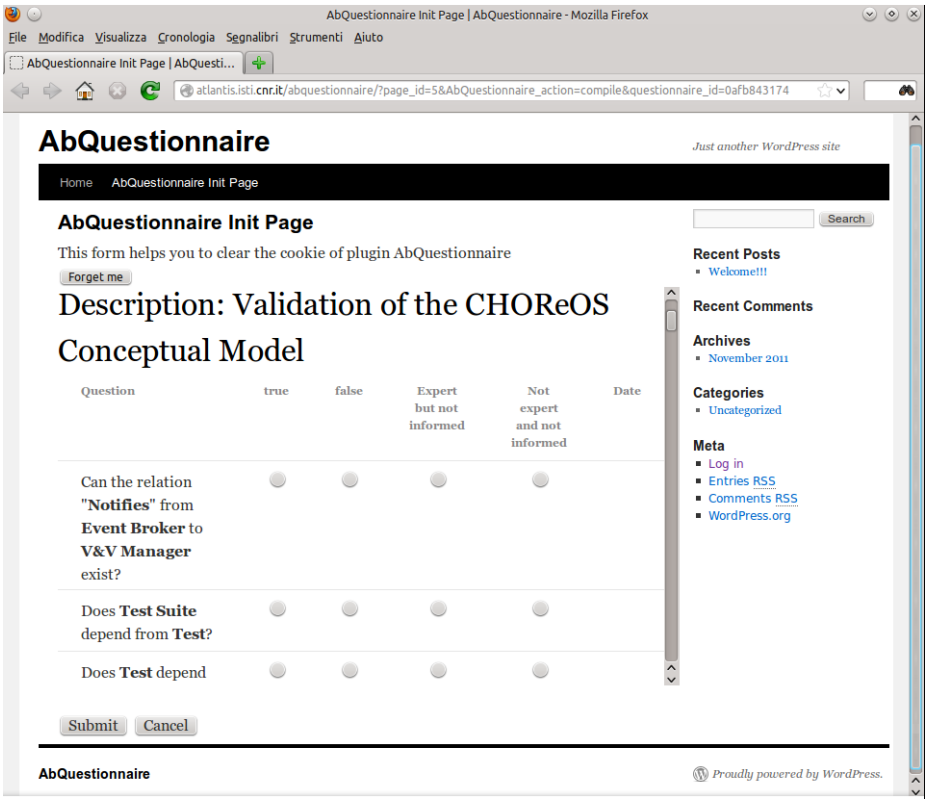


Figure 2.4: abQuestionnaire

Specifically, abQuestionnaire is a plugin for Wordpress (see at `http://wordpress.com/`). Like most web-based applications, it is structured into two main components: a back-end and a front-end. In the back-end component, registered users can administrate the questionnaires generated by MOTHIA. Specifically, the plugin provides canonical functionalities such as uploading, publishing, and unpublishing a questionnaire. In addition, it also supports for inviting the interviewed persons and reporting useful statistics about the published questionnaires. After the publishing phase, the front-end of abQuestionnaire allows invited persons (i.e., the DEs) to fill their questionnaires. The implementation of abQuestionnaire we used for distributing and collecting feedback from DEs is available at: `http://labsedc.isti.cnr.it/tools/abquestionnaire`. Fig. 2.4 shows a screenshot of the front-end on a generated questionnaire.

2.3. Distributed Questionnaires and Collected Answers

We used MOTHIA to generate 18 questionnaires. This number corresponds to the number of DEs from different CHOReOS partners having both effort and expertise for validating the CHOReOS domain model on behalf of the whole project. Each questionnaire covered all the areas of concern in the domain model, and included both deductive and distractor questions.

As said, MOTHIA produces NL questions requiring `Yes/No` answers. However, for the purpose of refining the CHOReOS conceptual model, each question produced by MOTHIA actually proposed the interviewees a multiple-choice answer based on four possible, and exclusive, items: (1) `Yes`, (2) `No`, (3) “I’m not an expert on this specific subject” (`NE`), and (4) “I’m an expert on the subject but I’m not sure what to answer” (`DK` - Don’t Know).

The rationale behind the scale we adopted is that, by means of both answers (1) and (2), we can compare the idea of a concept that a DE would like to formalize in the domain model (i.e., its properties, relations with other concepts, etc) with the actual representation it had in the model. The option (3) has

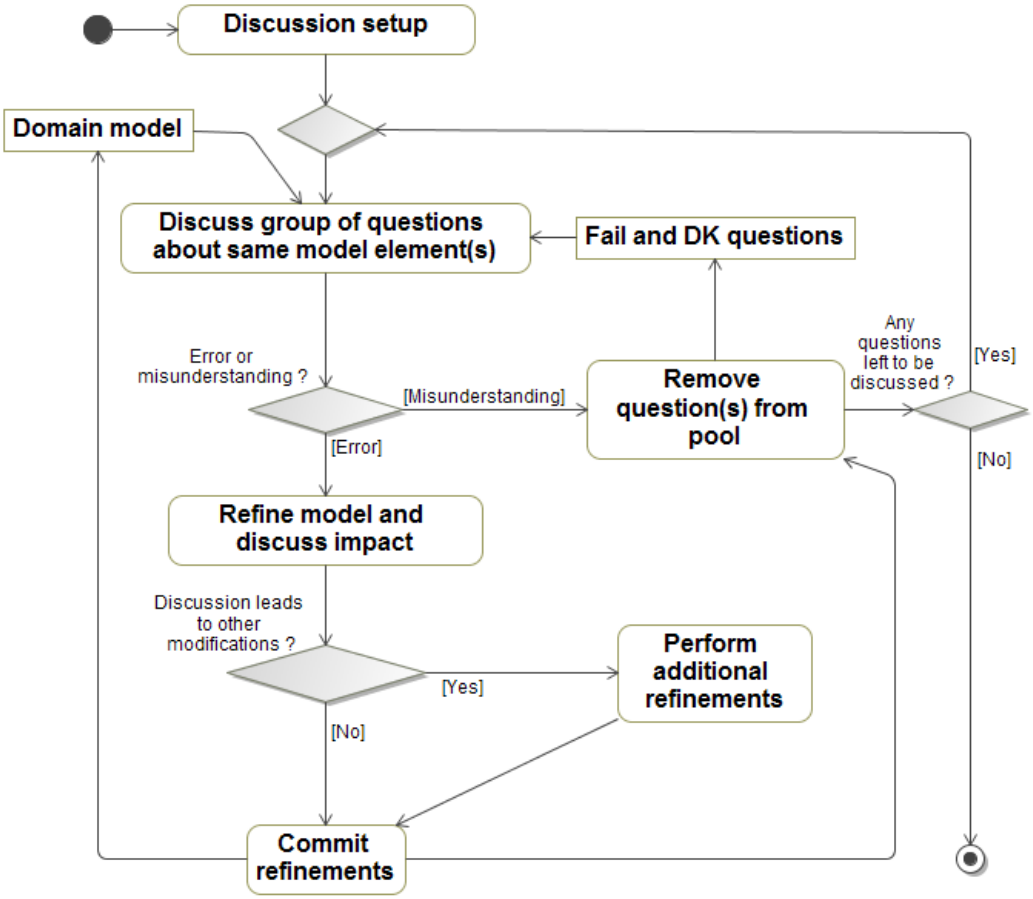


Figure 2.5: Model refinement activities performed with the questionnaires interviewees

been introduced because the questions are randomly generated from the domain model (i.e., to reduce bias) and there may happen that an interviewee is not able to give any feedback. Rather, the option (4) results quite useful for dealing with distractor questions, as they may stimulate the discussion of aspects that were not directly included in the domain model. It is useful to recall from Section 2.2.1 that a distractor question is created from false predicates on the input model (e.g., a non-existent relationship among two or more entities).

In previous experiences [1], MOTHIA was used only to deal with structural aspects of domain models, as expressed by UML Class Diagrams. In CHOReOS we also exploited the new features provided by MOTHIA supporting behavioral aspects based on both Activity and Use Case Diagrams. The implementation of both the patterns and the criteria we adopted are available at `http://labsedc.isti.cnr.it/tools/mothia`. The questionnaire was submitted to the DEs by means of the web-based supporting tool that was described in Section 2.2.3.

2.4. Correcting the Model from the Experts' Feedback

As previously mentioned, the CHOReOS domain model has been refined by considering the feedback provided by DEs when answering the questionnaires. Such a refinement process corresponds to the sub-process *Domain Model Refinement* of the overall methodology shown in Fig. 2.2.

The refinement process is shown in Fig. 2.5. A phone call was setup with each DE, who gave answers that differed from the expected ones (categorized as `Fail` or `DK`). The discussion was focused on groups of questions, manually clustered according to the domain elements involved. The discussion started by asking feedback on the given answers, in order to identify whether the DE had misunderstood

the question. If a real error was identified, its causes were investigated and discussed. The domain model was then refined accordingly. Interestingly, in some cases an error triggered modifications of elements that were not included in the original question(s). The discussion then continued with the next group of questions, until all the questions were analysed.

In line with the overall considerations on the validation of a system context given in [13], we identified four causes behind the most common errors revealed during the refinement phase. Specifically:

- *tacit assumptions*: most of the relevant elements of the domain model were modeled, but some aspects were not adequately considered. For example, the DEs did not mention an *evident* relationship among some CHOReOS domain elements, or a *major* property of one of them, giving them for granted.
- *incorrect information*: the representation of some elements was based on incorrect information about the domain. Specifically, the MEs made wrong assumptions about the concepts initially presented by the DEs.
- *semantic ambiguities*: the DEs underestimated the potential misunderstanding of the semantic of the terms used during the discussion with the MEs. For example, using either synonymous or improper terms resulting similar although actually denoting different domain elements.
- *unstated features*: one or multiple aspects of the domain model were overlooked in the preliminary version of the conceptual model in Deliverable D1.2. For example, the DEs initially preferred to postpone their inclusion (e.g., to simplify the domain model), but then these aspects were left unstated.

In a few cases, the DEs did not understand well the question and after the discussion, they amended the previous given answer by recognizing that the model was correct and they gave the wrong answer.

In the remaining of the section we report some representative cases fitting in the above classification. For each of them, we also present the refinements we performed on the CHOReOS domain model.

2.4.1. Tacit Assumptions

The first example about the identification of a tacit assumption deals with the *Composition Concern* detailed in Section 3.1.

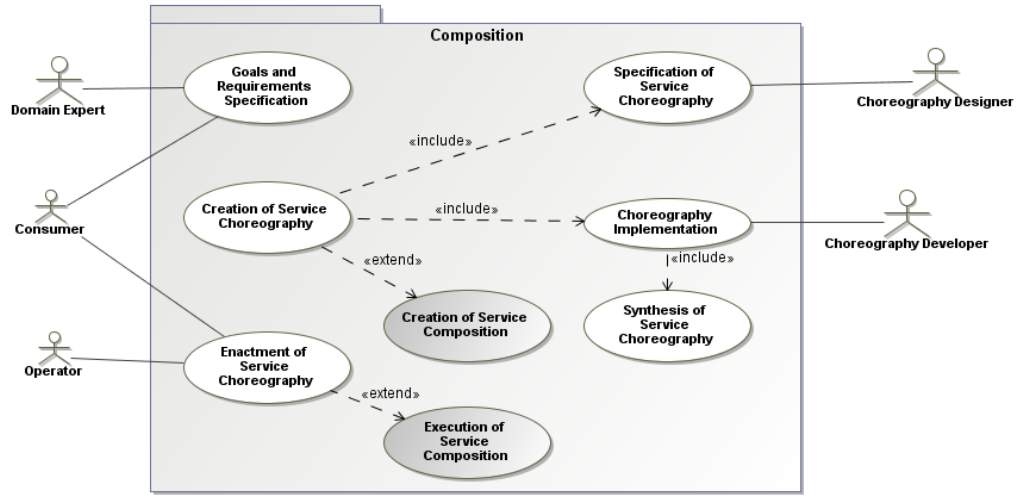
Specifically, the discussion with the DE led to identify that the actor `Choreography Developer` could be involved in the use case `Creation of Service Choreography`. This involvement relation was not included in the initial version the conceptual model as reported in Fig. 2.6(a). The DE asserted that this relation was intuitively assumed, and therefore it was not mentioned. The ME did not catch this information, so he/she did not include it in the domain model. Fig. 2.6(b) depicts the refined version of such a diagram.

2.4.2. Incorrect Information

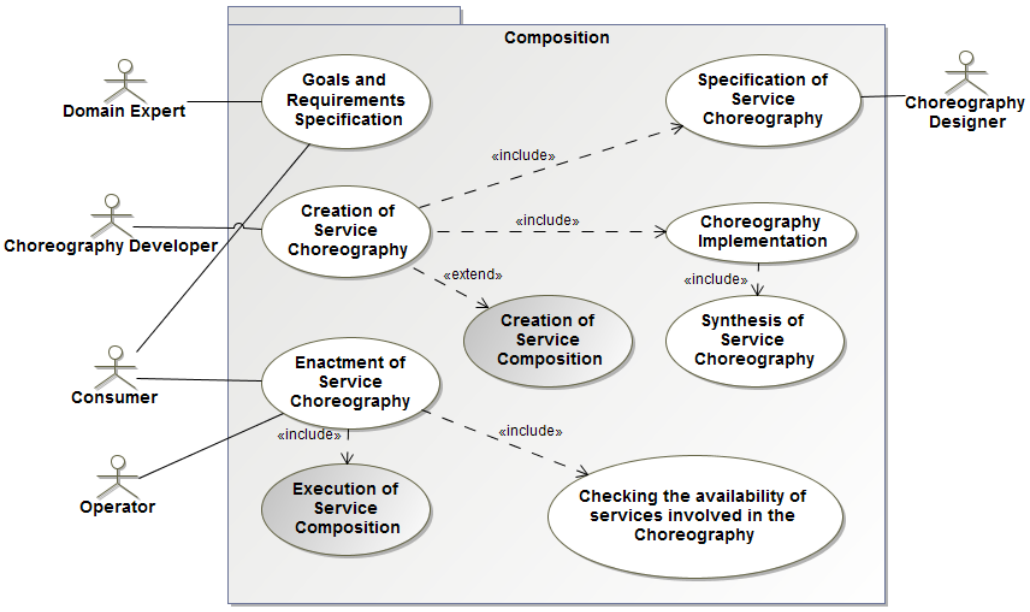
An example of wrong assumption made by MEs can also be found in the old *Composition Concern* specification in Fig. 2.6(a).

During the definition of the preliminary version of the domain model the ME kept the `Enactment of Service Choreography` as an independent behavior that could be used to augment the definition of the `Execution of Service Composition` use case. Nevertheless, a positive answer to the distractor question:

> Is the use case `Execution of Service Composition` a part of the use case `Enactment of Service Choreography`? (Q1)



(a) Initial model.



(b) Refined model.

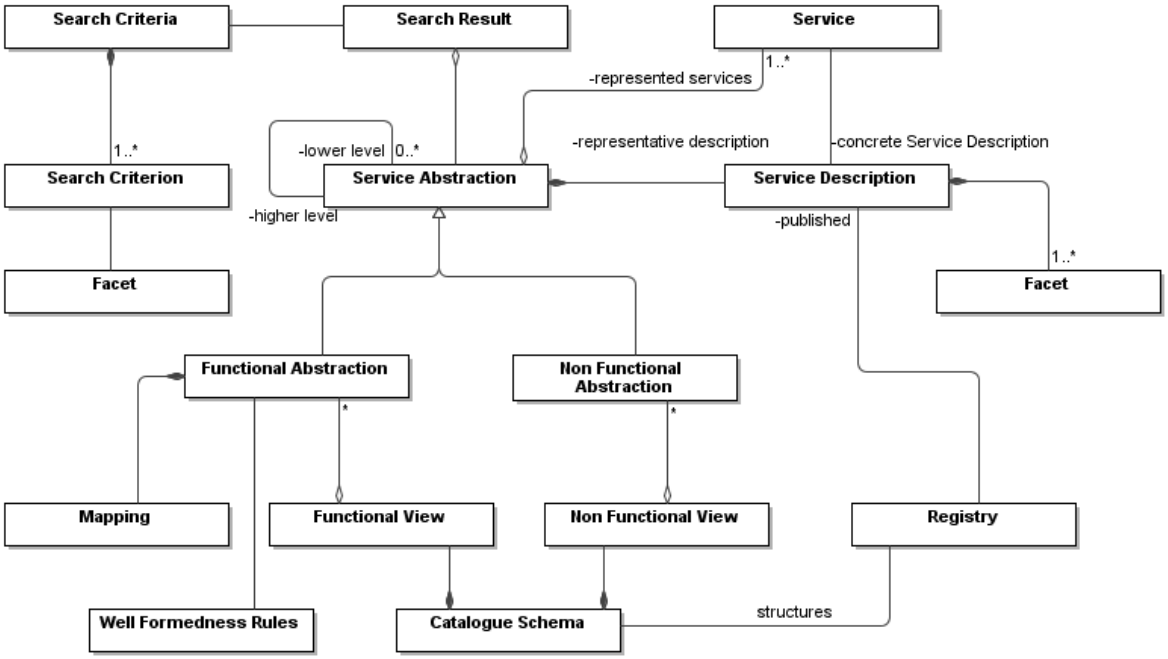
Figure 2.6: The Composition Concern

revealed the DE perception that the enactment of a choreography includes part of the behavior modeled by the execution of a generic service composition. As a consequence, the domain model was refined and the association connecting the two use cases was updated.

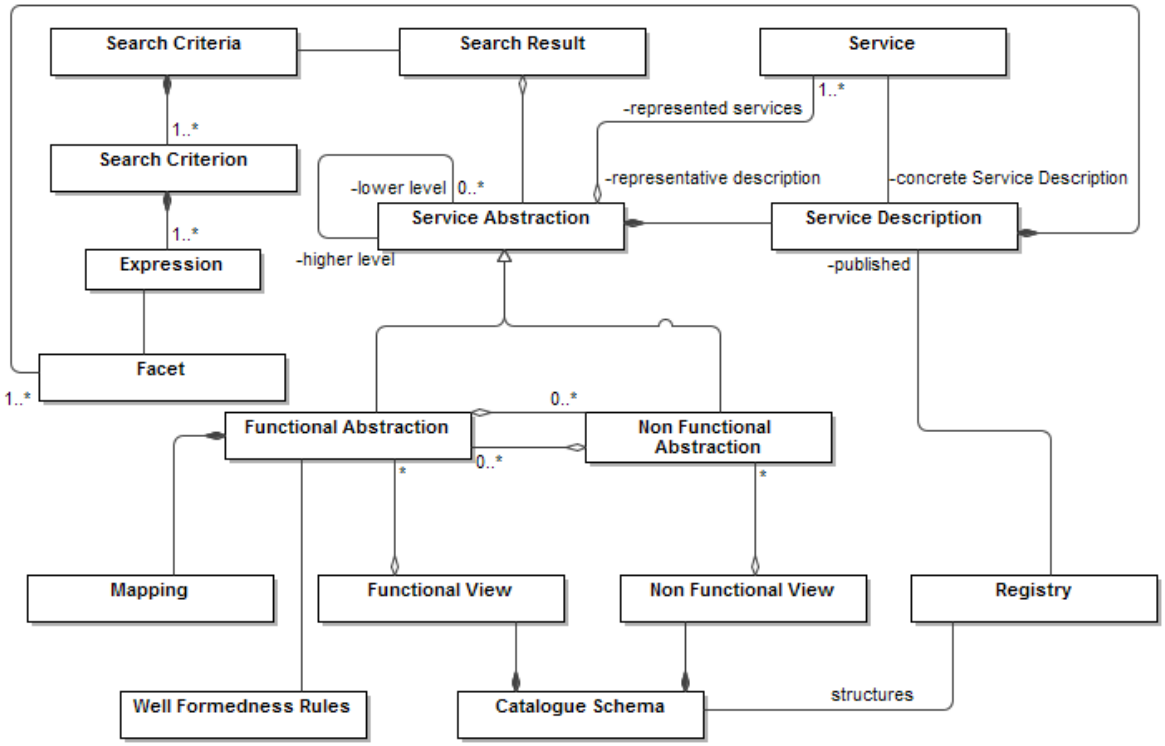
Another example about this category of faults refers to the *Analysis Concern*, and in particular to the test activation functionality. Specifically, the model in Fig.2.8(a) has been refined by discussing the question:

> Is `Verification Event` always the previous action executed before the action `Activation Evaluation`? (Q2)

The answer to Question Q2 given by the DE was `true` since the selection and the execution of a test can be done only after the verification of the event triggering the test activation. However, the expected answer computed by MOTHIA was `false` because of an inaccuracy reported in the model. In particular, the initial node of the model has a direct link to the activity `Activity Evaluation`. Thus, such activity may be performed first, even before the starting `Verification Event`. The ME

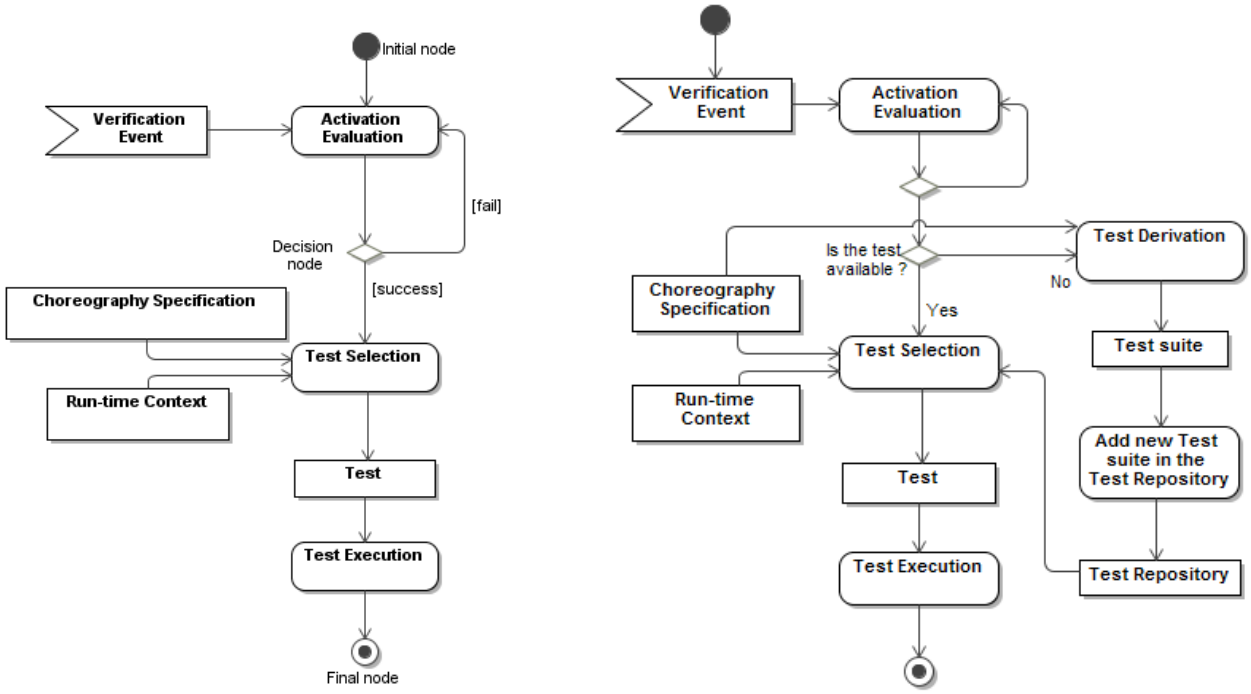


(a) Initial model.



(b) Refined model.

Figure 2.7: The Discovery Concern



(a) Initial model.

(b) Refined model.

Figure 2.8: The Test Activation Functionality

represented this scenario as he/she unilaterally assumed that the activation of test functionalities not associated to any specific event was also admissible. This required modifying the model by changing the connection of the initial node, which has to point to `Verification Event` (see Fig. 2.8(b)).

Note that the adoption of a single incoming and outgoing flow to/from an action is strongly recommended, showing all joins and merges explicitly [3]. In this sense, the refined model resulted even more robust to possible semantic misunderstanding.

2.4.3. Semantic Ambiguities

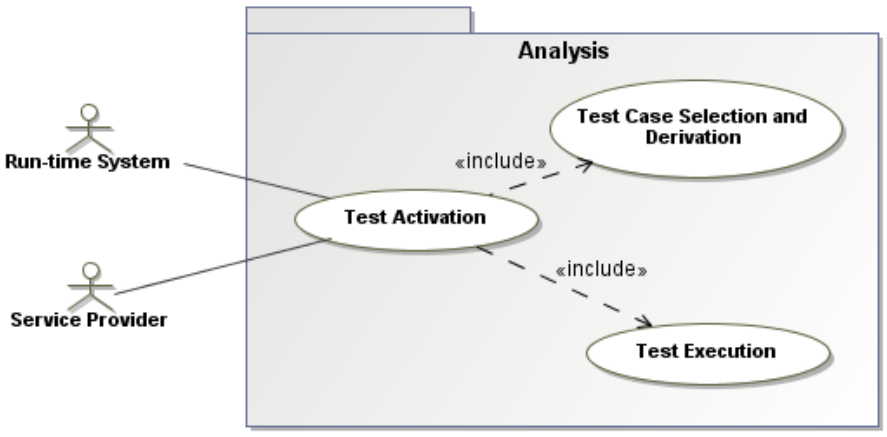
Semantic ambiguities are the most common and difficult kind of error that may arise from misleading communications between DEs and MEs. As deeply investigated in the field of requirement engineering [13], they are often due to an awkward use of either synonymous or terms that look similar although denoting different concepts.

Considering both the wideness and the difficulty of the problem [13], MOTHIA does not intend to provide neither a general nor a definitive solution to it. Nevertheless, in CHOReOS we found that the open discussions we had with DEs during the validation contributed to highlight some of those problems. For instance, with respect to the source model in Fig. 2.7(a), an unexpected answer was received for the question:

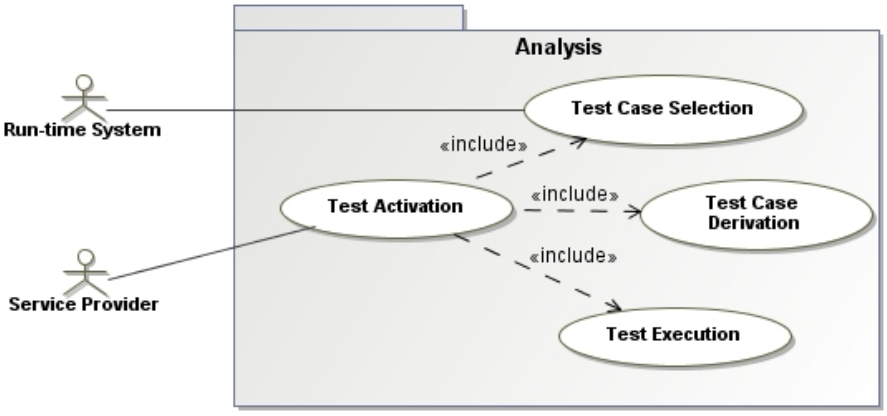
> Can a relation from `Service Description` to an unlimited number of `Facet` exist? (Q3)

After investigating the domain model, we realized that the MEs defined two entities called `Facet` (located in two different packages of the domain model), expressing two different concepts. Note that in the model of Fig. 2.7(a) the two entities appear close in the same diagram only for the sake of presentation.

When generating the Question Q3, MOTHIA considered the left-side element called `Facet`, thus the expected answer it computed was `false`. Instead, answering the question, the DEs assumed that the



(a) Initial model.



(b) Refined model.

Figure 2.9: The Testing Activities

queried element was the rightmost one. Even more, during the discussion with the DEs, it was clear that the two concepts were actually describing the same abstract entity. Consequently, they have been unified in the refined version of the domain model (see Fig. 2.7(b)).

2.4.4. Unstated Features

The last category of errors we discovered are mainly due to some preliminary simplification of the domain model that then led to unstated features. For most of these errors, there were no questions generated by MOTHIA that supported their direct identification. In fact, most of the concepts fitting this category were not included in the model at all.

Nevertheless, the methodology presented in this chapter together with the systematic generation of questionnaires by MOTHIA gave the opportunity to both MEs and DEs to start open discussions about the produced artifacts. Such discussions were guided by the investigation of those elements in the model requiring further considerations, i.e., the ones in questions where the expected answer was different from the given one. In such a way we were able to reveal some of the unstated features. As an example, tackling the incorrect information included in the model at Fig. 2.6(a) (see Section 2.4.2), the discussion led to realize that the enactment of a service choreography (as modeled) may implicitly assume that all the services are already available. Since this is not true in the general case, the DE preferred to explicitly model a separated use case dedicated to the verification of the availability of all the required services as part of the choreography enactment (see Fig. 2.6(b)).

Similarly, the discussion with the DE about the incorrect information revealed by Question Q2, pointed out some other issues in the model depicted in Fig. 2.8(a). In fact, the DE completely omitted the specification of a decision branch about the derivation of new test suites, which can be required to test a given choreography specification. Thus, the corrected model in Fig. 2.8(b) has been produced by adding the new activities `Test Derivation` and `Add new Test suite in the Test Repository`, and their connections with the rest of the diagram elements. Given a choreography specification to be tested, the main idea is to check if the required test suite is available in test repository; If not, derive the test suite and add it to the repository. The new test suite can then be executed.

Interestingly, the discussion on the models in Fig. 2.8 also led to a refinement of the model in Fig. 2.9, in order to explicitly represent the `Test Case Derivation` functionality. Furthermore, the actor `Run-time System` was modified by removing his relation with the `Test Activation` use case (which is performed only by the `Service Provider`), and by adding the relation with the use case `Test Case Selection`.

3 Final CHOReOS Conceptual Model

This section describes the final version of the CHOReOS conceptual model. Figure 3.1 shows the concerns of the conceptual model after the refinement process was applied to the first release reported in Figure 2.1.

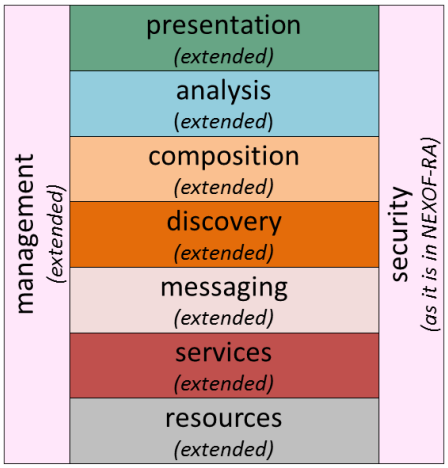


Figure 3.1: Concerns of the CHOReOS conceptual model after the refinement activities

Concern	D1.2	D1.4a
Composition	Extended from NEXOF-RA	Refined from D1.2
Discovery	Extended from NEXOF-RA	Refined from D1.2
Analysis	Extended from NEXOF-RA	Refined from D1.2
Messaging	Extended from NEXOF-RA	Extended from NEXOF-RA
Services	As it is from NEXOF-RA	Extended from NEXOF-RA
Resources	As it is from NEXOF-RA	Extended from NEXOF-RA
Management	Extended from NEXOF-RA	Extended from NEXOF-RA
Presentation	As it is from NEXOF-RA	Extended from NEXOF-RA
Security	As it is from NEXOF-RA	As it is from NEXOF-RA

Table 3.1: Overview of the changes performed on the CHOReOS conceptual model

More precisely, as shown in Table 3.1, the final conceptual model includes both the extensions/refinements of the model already defined in Deliverable D1.2 (i.e., *Composition*, *Discovery*, and *Analysis*), and the additional extensions/refinements that were missing in the previous version of the model (i.e., *Services*, *Resources*, and *Presentation*). Note that, for the *Management* and *Messaging* concern, no further refinements were needed with respect to the previous version. Moreover, note also that after the refinement process, the *Security* concern of NEXOF-RA is the only one kept as it is, without adding specific CHOReOS extensions. In fact, as also clarified in Deliverable D1.4(b) [18], the study of technologies dedicated to enforcing security, privacy, and trust is beyond the scope of the CHOReOS project; instead, state of the art technologies and possibly latest results from projects focused on secu-

rity solutions are built upon for the development of CHOReOS use cases -if and when needed-.

In the remainder of this section, the final CHOReOS conceptual model is detailed. Its complete specification, defined with the MagicDraw modeling tool, is available for download at `http://www.choreos.eu/bon/Download/Deliverables`. Since some diagrams were previously shown to introduce the refinement process we have adopted, in this chapter we do not duplicate them, rather we refer to the relevant diagrams when appropriate.

3.1. Composition Concern

This concern is related to the concepts of *Orchestration* and *Choreography* of services as specializations of the service composition concept. The functionalities defined in this concern, after being refined, are shown in Figure 2.6(b). In particular, a *Choreography Designer* can *specify* a service choreography with respect to the *Goal and Requirement Specifications* provided by the *Domain Expert* and *Consumers*.

The *Creation of a Service Choreography* encompasses its specification and its implementation. The implementation of a service choreography includes the *synthesis* of coordination delegates for suitably coordinating the involved services. Finally, the *Operator* has the responsibility of enabling the *execution* of the created choreography, which can then be *enacted* by the consumer as needed.

3.1.1. Structure View

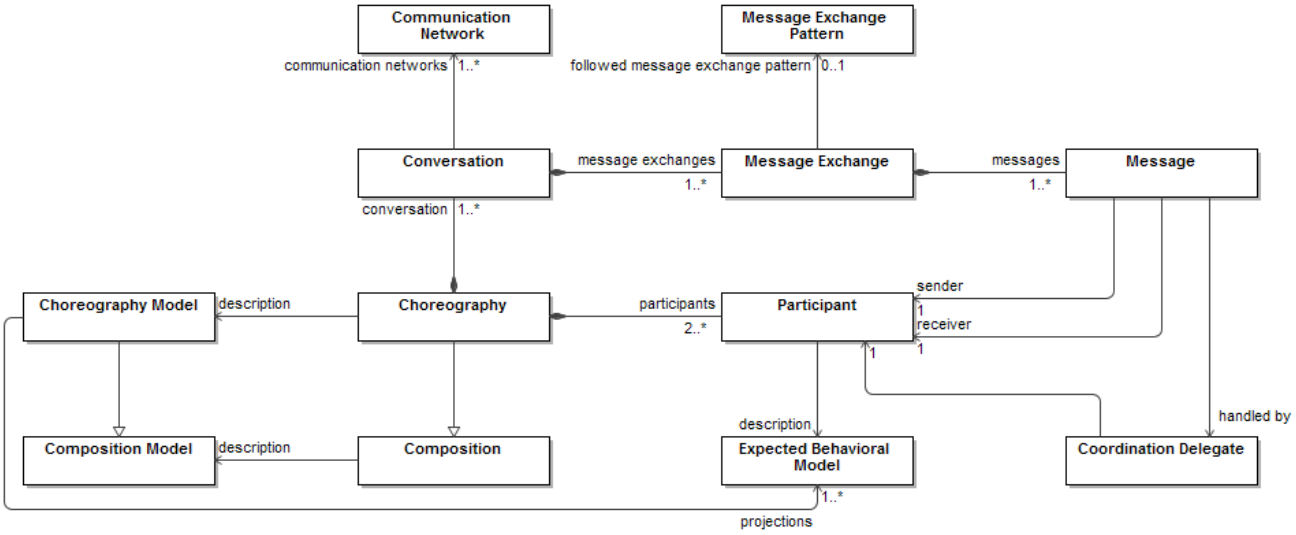


Figure 3.2: Structure View of the Composition Concern

Figure 3.2 depicts the refined structure view of the composition concern, showing only the extensions related to the *Choreography* concept. A *Choreography*, as described by a *Choreography Model* (specializing the *Composition Model*), is related to (i) the ability to model business contracts, i.e., *Conversations* in terms of a set of *Messages* which can be exchanged (possibly according to a *Message Exchange Pattern*) among multiple *Participants* and (ii) how a business contract is "enforced" with respect to the individual *Expected Behavioral Model* of participants. For a given participant, the *Expected Behavioral Model* specify the expected interaction behavior (in terms of allowed sequence of *Messages*) as projected from the *Choreography Model*. In summary, a choreography specifies how messages are exchanged among participants in a global conversation. Thus, the focus is not on orchestrating the work performed within each participant, but rather on the external exchange of information (messages)

between these participants. In this respect, *Coordination Delegates* play a key role: they are additional software entities, which are interposed among participant services in order to coordinate the services' interaction in a way that the resulting collaboration realizes the specified choreography.

3.1.2. Data Flow

Building upon the previous discussion, a choreography can be seen as a collaborative process among interacting participants. The message exchange globally defined by the choreography has to be projected among different participants according to the local roles they play to fulfill the global choreography. In the following, the functionalities *Specification of Service Choreography*, *Goal and Requirement Specification*, *Synthesis of Service Choreography*, and *Enactment and Execution of Service Choreography* are described. The *Discovery of Service* is then described in Section 3.2.

Specification of Service Choreography As shown in Figure 3.3, *Specification of Service Choreography* takes as input *Choreography Patterns* as derived from the specification of the choreography goal and requirements (see Figure 3.4). Then, the service choreography specification process produces a *Choreography Model*. Such a model is taken as input by a service choreography synthesizer together with the service abstractions. The latter describes the abstract services that can be potentially used to realize the choreography.

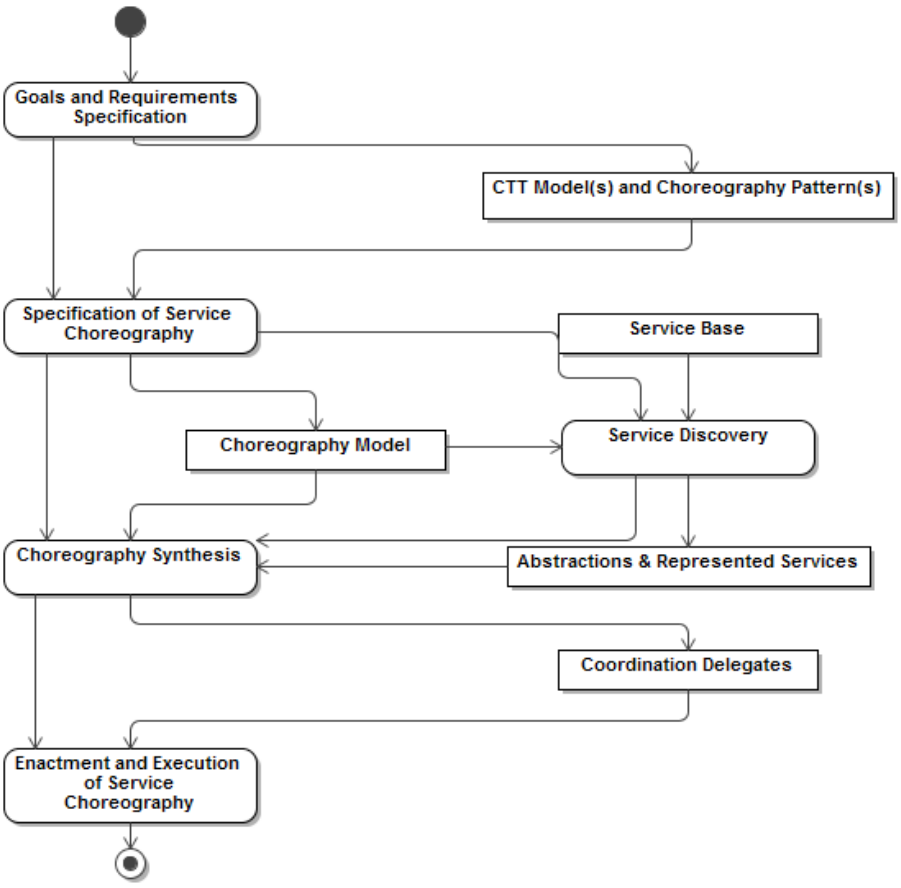


Figure 3.3: Data Flow in the Composition Concern

Such abstract services have been discovered from the *Service Base(s)* (see Section 3.2) with respect to the specification given in the *Choreography Model*. As further detailed in the following, the output of the service choreography synthesis is a set of *Coordination Delegates*, which are generated

to support choreography enactment in a distributed way, hence enabling the choreography realization enforcement. In the overall process, MDE techniques (employing model transformations and generative techniques) help to deal with some FI characteristics. For instance, the ultra-large scale of the FI is addressed by supporting the systematic automation of large-scale choreographies distribution and the automatic deployment of a possible large number of coordination delegate instances.

Goal and Requirements Specification As shown in Figure 3.4, requirements are expressed by *Domain Experts* (domain specific requirements) and *Consumers* (user needs) using a structured approach that includes qualitative qualifiers, satisfaction ratings, and prioritization. The domain expert oversees the process, receiving the consumer requirements either directly or indirectly through the *Evolving Requirements Specification*. The resulting set of prioritized quality-based requirements is then associated with choreography strategies, which are expressed in the form of patterns. The service choreography patterns encapsulate different complex choreography decisions made in the presence of user requirements. Each provides a structure for associating user requirements with different possible choreographies of classes of services (i.e., groups of similar services as discussed in Section 3.2) that, when implemented, deliver service-oriented applications with different qualities. Each pattern considers the qualities of the services to be invoked, the qualities that different choreographies offer and how, when combined, these service and choreography qualities deliver systems of different qualities. Users of CHOReOS solutions will match their user requirements to the service choreography patterns that will inform more detailed choreography design.

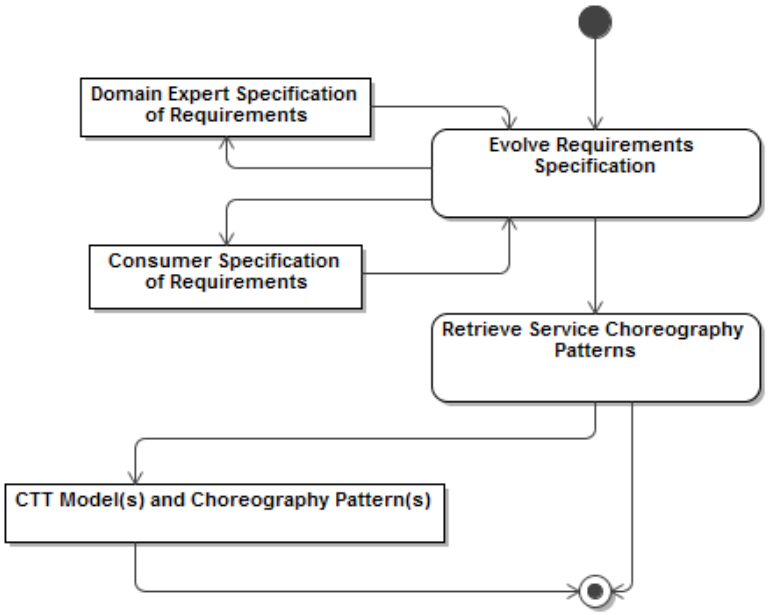


Figure 3.4: Goal and Requirements Specification of the Composition Concern

Importantly, the goal and requirement specification techniques will support scalability, as testified by the adoption of i* [4], which has previously been applied to model *large-scale systems* [9].

Synthesis of Service Choreography As already anticipated, in CHOReOS we are tackling the problem of *automatic realizability enforcement* that concerns the realization of service choreographies by reusing (third-party) services.

That is, with reference to Figure 3.5, given a choreography specification and a set of existing services discovered as suitable participants (by exploiting the service abstractions described in Section 3.2), we restrict the interaction among them so as to fulfill the collaboration prescribed by the choreography

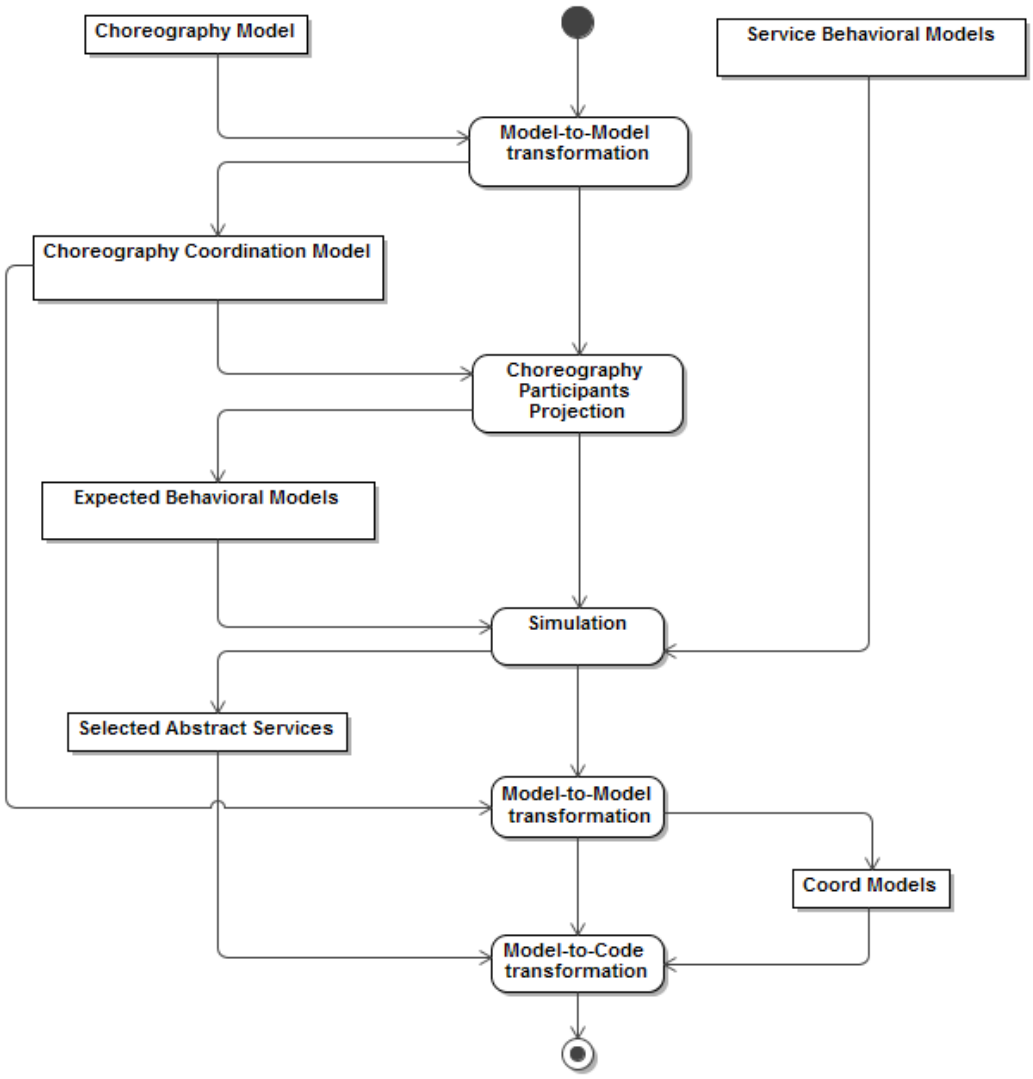


Figure 3.5: Synthesis of Service Choreography of the Composition Concern

specification. This requires to extract from the *Choreography Model* the global coordination logic to be then distributed and enforced between the participants. The coordination logic in extracted from the choreography model by a *Model-to-Model Transformation* and it is codified into a *Choreography Coordination Model*. A detailed description of the service abstractions and related mappings to concrete services, together with that of the coordination model, can be found in Deliverable D1.4(b) [18]. To do this, we consider the local behavior of each choreography participant (*Expected Behavioral Models*), as extracted from the *Choreography Model* by the *Choreography Participants Projection* activity. Then, by exploiting the *Simulation* activity, the model of the end-point projection of each participant is used to select those services (*Selected Services*) whose behavior simulates the role to be played by the participant. The service behavior of the discovered services is returned by the discovery activity as part of the service descriptions (see Figure 3.3). However, although services have been selected as suitable end-points to realize all the participant roles of the specified choreography, their composite interaction may prevent the choreography realizability if left uncontrolled (or coordinated in a wrong way).

That is, as detailed in the companion Deliverable D1.4(b) [18], the notion of coordination protocol becomes crucial. In fact, it might be the case that the collaborating services, although potentially suitable in isolation, when interacting together can lead to *undesired interactions*. These are interactions that do not belong to the set of interactions modeled by the choreography specification and can happen when

the services collaborate in an uncontrolled way. To prevent undesired interactions, the CHOReOS platform automatically synthesizes additional software entities, called *Coordination Delegates* (CDs) that, being interposed among the participant services, coordinate the services' interaction in a way that the resulting collaboration realizes the specified choreography. The CDs are automatically derived by a *Model-to-Code transformation* and exploit (local) coordination information codified into the so called *Coordination Models*. The latter are automatically generated out of the *Choreography Coordination Model* through a dedicated *Model-to-Model Transformation*.

Enactment and Execution of Service Choreography Generated coordination delegates are deployed in the CHOReOS execution environment providing the required resources to collaboratively execute the specified choreography. In this respect, the *Resource concern* described in Section 3.6 introduces the CHOReOS implementation and refinement of the NEXOF-RA *virtualized infrastructure service* supporting the execution of choreographies.

3.2. Discovery Concern

In CHOReOS we go beyond NEXOF-RA by introducing new high-level concepts in the discovery concern, along with corresponding lower-level concepts in the structural and the data-flow views of this concern. Specifically, our contribution is summarized in the following points:

- A new high-level concept is introduced in the discovery concern, namely the *Abstractions Recovery* functionality. The purpose of this functionality is to reverse engineer *service abstractions*, i.e., groups of available services that are characterized by similar functional/non-functional properties (i.e., facets in NEXOF-RA terminology), out of services that become available over time. As detailed in the rest of this subsection, service abstractions are hierarchically structured.
- Moreover, the concept of service abstractions is employed so as to facilitate the CHOReOS browsing and searching functionalities of the service discovery concern. In particular, as showed in Figure 3.8, *browsing* and *searching* is based on descriptions of *service abstractions*, instead of descriptions of services, so as to enable the consumers to efficiently and effectively retrieve multiple candidates of services that offer similar properties.

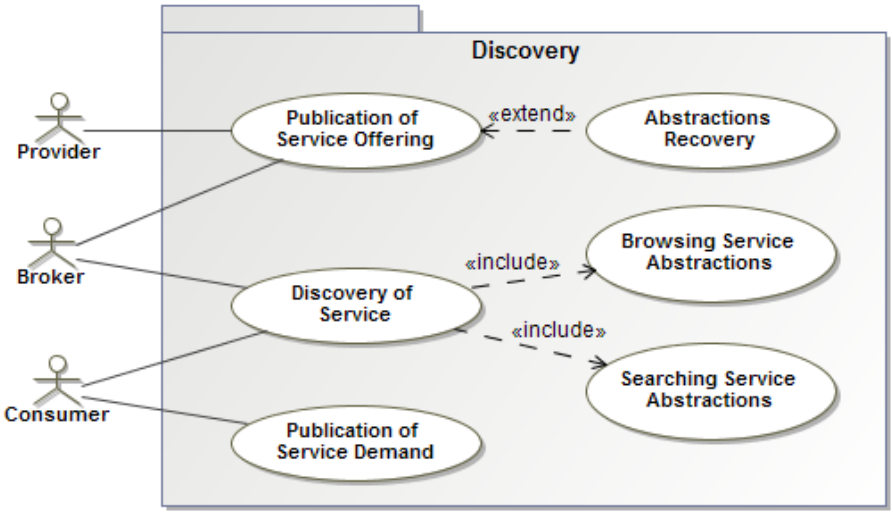


Figure 3.6: Discovery functionalities

3.2.1. Structure View

The new structure view of the discovery concern is shown in Figure 2.7(b). The *Abstractions Recovery* functionality, along with the concept of service abstractions are the central concepts that affect the definition of the CHOReOS information model of the service discovery concern:

- As showed in Figure 2.7(b), a *service abstraction* consists of a set of grouped *services*. Each service is associated with a corresponding *service description*.
- A service abstraction is further associated with an abstract service description. The abstract service description comprises a set of *facets* that represent the common/similar facets of the grouped services.
- A service abstraction may be associated with other lower-level service abstractions that represent groups of services, which offer more (and possibly more concrete) functionalities. Moreover, a service abstraction may be associated with higher-level service abstractions that represent groups of services, which offer fewer and possibly more abstract functionalities.

We distinguish the following different types of abstractions:

- *Functional abstractions:* The main purpose of a functional abstraction is to represent a group of services that offer similar functional properties (i.e., functional facets). At a glance, the description of a functional abstraction is characterized by a set of abstract functional facets, such that there is *mapping* between the abstract facets and the concrete functional facets of the services that are represented by the functional abstraction. To facilitate choreography adaptation and specifically the substitution of the services used in a choreography, the mapping must preserve certain *well-formedness rules* of behavioral sub-typing [8], adapted to the specificities of service-orientation. The definition of relationships between higher and lower level abstractions should also be based on such rules. A functional abstraction may be associated with non-functional abstractions, defined in the following.
- *Non-functional abstractions:* The main purpose of a non-functional abstraction is to represent a group of services that are characterized by similar non-functional property(ies) (i.e., non-functional facets) such as performance, reliability, availability, reputation, cost. Hence, the description of a non-functional abstraction is characterized by abstract non-functional facets that reflect the range of the concrete non-functional facets of the grouped services.

Based on the aforementioned functional and non-functional abstractions, the *catalogue schema* of a service registry (i.e., *Service Base* in CHOReOS terms) can be organized in terms of different *functional* and *non-functional views* that can be browsed or searched with respect to given service demands. The *results* of the browsing/searching are groups of services that come along with the service abstractions that represent them.

3.2.2. Data Flow

In this section all the functionalities of the *Discovery* concern are presented singularly.

Publication of Service Offer: The publication of service offer functionality allows the advertisement of information about a service provided as input by a corresponding service provider (Figure 3.7).

The *subscription* activity may further trigger the *abstraction recovery* functionality depending on whether a set of published services can be categorized in already existing service abstractions or not. In the latter case, the *functional* and/or the *non-functional* views of the registry should be updated via the execution of the abstraction recovery functionality.

A service provider may further request to *change*, or *remove* a particular service offer that was previously published. These requests may trigger the need to *update* the content of a service abstraction included in the functional and/or the non-functional views of the service base.

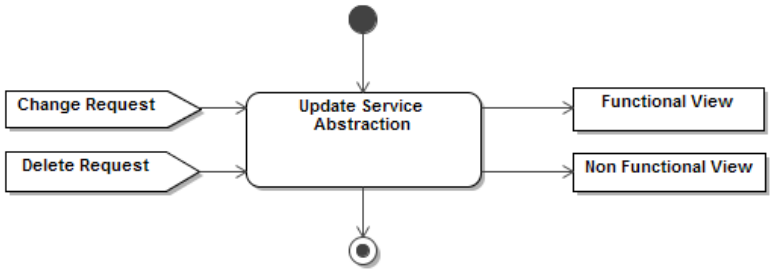


Figure 3.7: Publication of Service Offer functionality

Abstraction Recovery: The abstraction recovery functionality may be periodically initiated or on-demand, depending on the availability of new services over time (Figure 3.8).

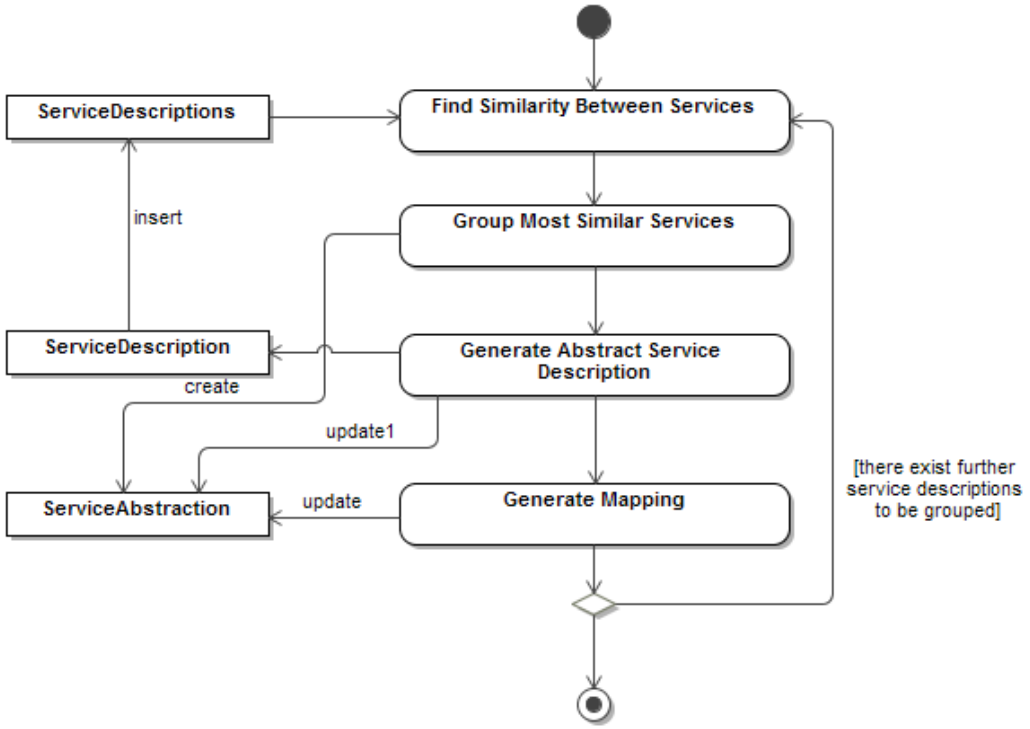


Figure 3.8: Abstraction Recovery functionality

The input to the functionality is a set of available service descriptions. Then, the basic activities involved in the recovery of service abstractions include:

- The calculation of similarities between the functional/non-functional properties of the available services.
- *Finding* and *grouping* of the most similar service descriptions.
- The *generation* of an *abstract service description* for each particular group of similar services.

- The *generation of mappings* between the properties of the abstract service description and the properties of the grouped services.
- The whole process is iterative; the abstract service description of each *service abstraction* produced by the recovery process is inserted in the input set of available service descriptions, so as to enable the creation of higher-level abstractions in subsequent iterations of the recovery process.

The similarity criteria and the metrics used for the recovery of service abstractions depend on whether the goal is to produce functional or non-functional abstractions.

Discovery of Service: As specified in the NEXOF-RA discovery concern, the discovery of services can be done either by browsing the content of the registry or automatically via algorithms that match a given service demand against the service descriptions stored in the registry. As previously mentioned, the concept of service abstractions facilitates the efficient and effective retrieval of multiple candidates of services that offer similar properties (Figure 3.9). More specifically:

- According to the CHOReOS *browsing for abstractions* activity, the consumer explores the abstractions hierarchies of the *functional* and *non-functional* views of the registry to locate groups of services that offer similar properties, which satisfy the consumers *service demand*. By construction, the abstractions must be such that, if the properties of a service abstraction satisfy the demand of the consumer, then any service represented by this abstraction may also satisfy the demands.
- The CHOReOS *searching for abstractions* activity, provides means to match the consumers demands against abstract service descriptions that characterize abstractions, which group services with similar properties. As also mentioned in the case of the browsing activity, the abstractions must be constructed such that if the matching of a service demand with a service abstraction description is successful, then any of the services that are represented by the matched service abstraction may be a candidate solution that satisfies the issued service demand.
- Then, the outcome of both of the aforementioned activities is a set of services that meet the required service demand; the group of discovered services comes along with the abstraction that represents the discovered services.

Publication of Service Demand: As shown in Figure 3.10, the publication of service demand is an asynchronous way for discovering services. Consumers register their interest for services along with a specific service demand specification. Then, whenever a new service offer is published, it is checked against the demands (based on the typical searching for service functionality) of the consumers and corresponding notifications are sent if the demands are satisfied by the new service offer. Obviously, the complexity of this particular functionality is proportional to the complexity of the typical searching for service functionality, due to the fact that the search functionality must be triggered for every consumer publication of service demand. The concept of service abstraction may be employed to reduce the aforementioned complexity. In particular, we adapt the publication of service demand functionality as follows.

- According to the subscription activity, a consumer provides as input a *service demand*.
- Based on the service demand, the *Searching For Service Abstraction* activity is triggered.
- If this activity results in a service abstraction that satisfies the service demand, then the *demand is associated* with this service abstraction.

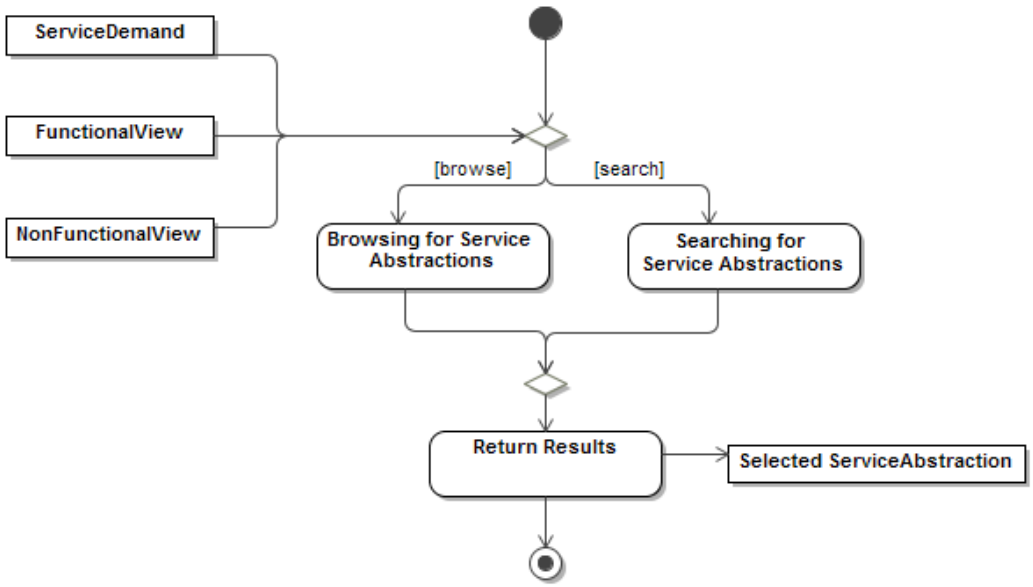


Figure 3.9: Discovery of Service functionality

- Subsequent publications of new service offers that result in new members of the service abstraction trigger the *retrieval of associated consumers*.
- *Notifications are sent* to the associated consumers, without the need for matching the service demands against the service offers.

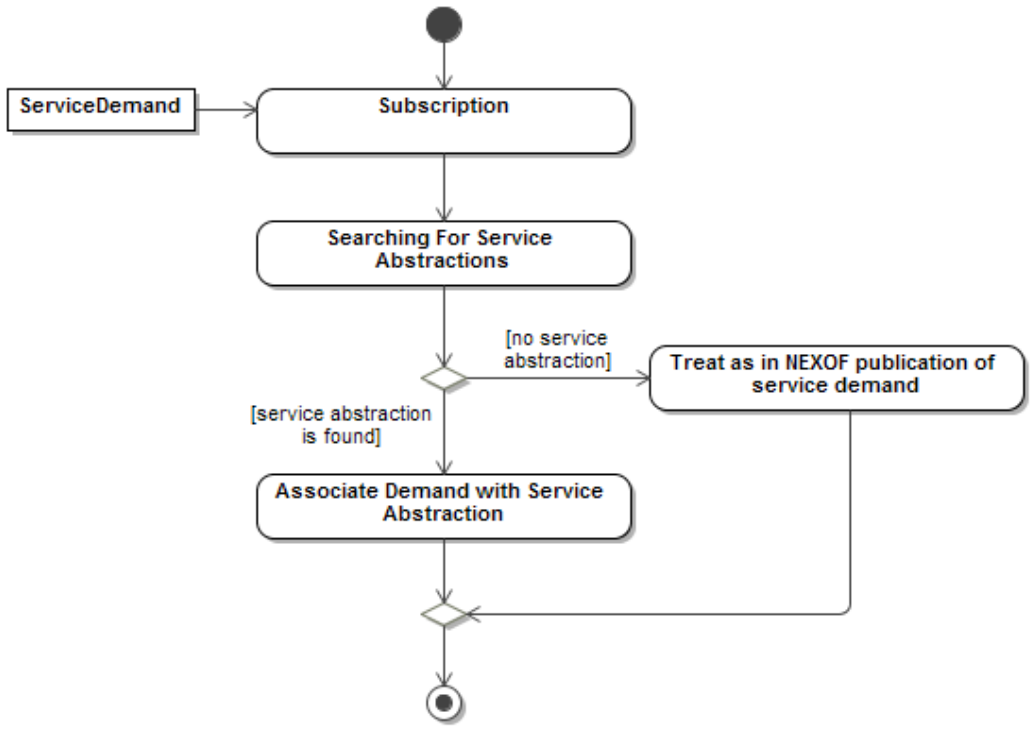


Figure 3.10: Publication of Service Demand functionality

3.3. Analysis Concern

In CHOReOS, the analysis concern encompasses (i) dependency analysis, (ii) QoS prediction, and (iii) testing activities (Figure 3.11). Either the choreography designer or the choreography developer performs the dependency analyses activities. Use cases in this domain include assessing the impact of changing roles in a choreography, changing services that are being used to fulfill choreography roles, and changing the choreography schema/diagram. In a similar vein, QoS prediction can be conducted by the aforementioned actors and involve predicting the QoS from choreographies that already have bound services. Finally, testing activities can be activated both by the Run-time System environment or by a Service Provider to test specific services. The automatic activation tries to provide a solution to the possibility that the run-time context may change. V&V activities can then provide a useful mean to assess if the change could lead to possible failures. The *Test Activation* use case includes *Test Case Selection and Derivation* and *Test Execution*.

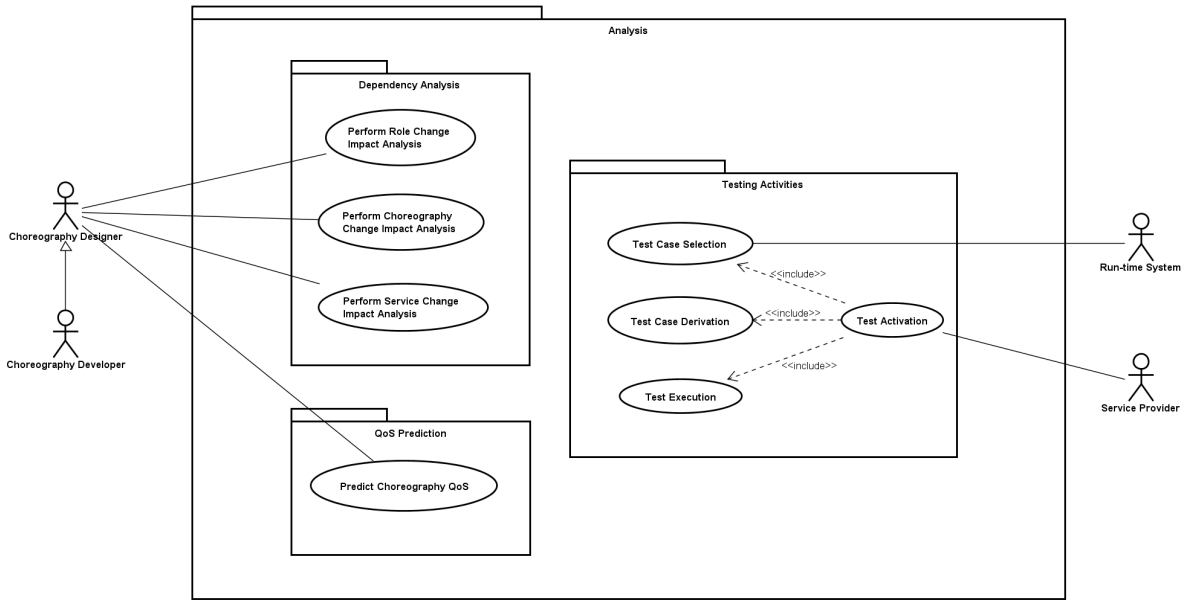


Figure 3.11: Analysis Functionalities

3.3.1. Structure View

Figure 3.12 depicts the structural view of the dependency analysis and QoS predictor concerns. The dependency analyzer relies on two important registries to accomplish its goals: the discovery registry and the choreography registry. In particular, the analyzer asks the discovery registry for the list of compatible services per choreography participant (role) and asks the choreography registry for the list of services implementing each role of the enacted choreographies. As the dependency analyzer, the predictor interacts with the discovery registry and the choreography registry. The predictor interacts with the discovery registry in order to have QoS values on concrete services part of choreographies. The interactions with the choreography registry and choreography model serves for understanding the global structure (services, participants, interactions between services) of choreographies. The linear solver is an external component specialized in the classical resolution of linear programs. Both the dependency analyzer and the QoS predictor rely on a general purpose graph manipulation and analysis toolkit, which provides fundamental algorithms and metrics for graphs. Finally, we highlight that a series of concepts from other concerns also take part in the domain model. For instance, the model includes the discovery registry and service description concepts from the Discovery Concern, as well as the choreography, message, and participant concepts from the Composition Concern.

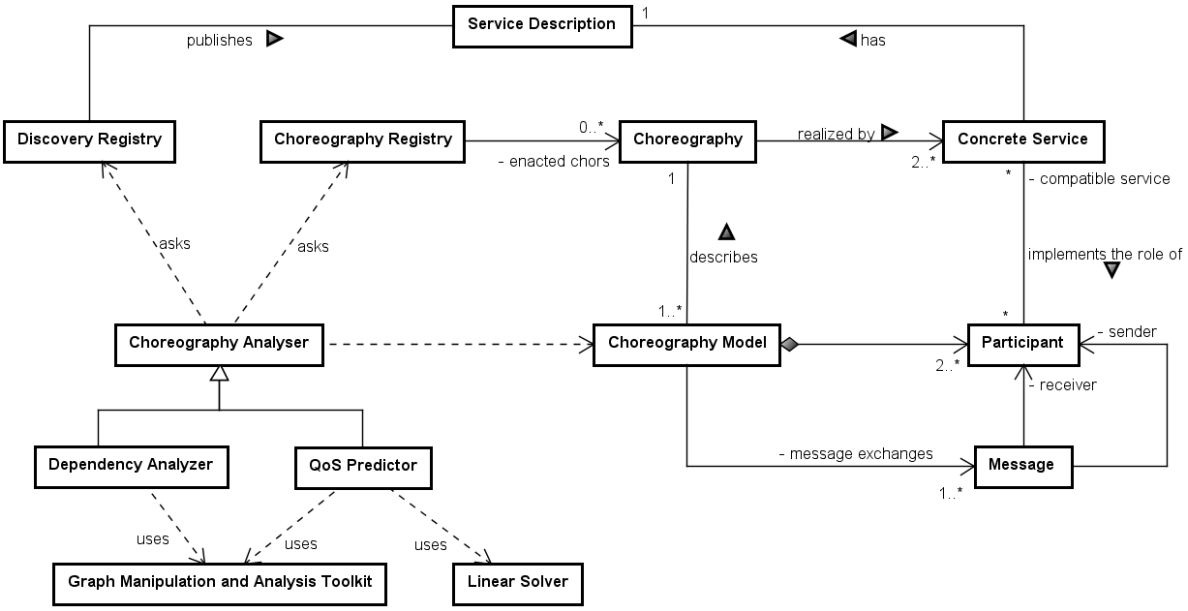


Figure 3.12: Structure View of the Dependency Analysis (Analysis Concern)

Figure 3.13 depicts the structural view of the testing activities. The following elements are especially relevant:

- *Test Selection Strategies*: this concept describes the algorithms that will be defined in order to derive test cases from choreography specifications, i.e., *Choreography Model*, and from information coming from the run-time environment (*Run-time Context*). This will make testing activities at run-time particularly effective and complementary to off-line testing activities, which cannot take into account real execution environment in the test derivation related activities.
- *Test Agent*: represents those elements that support the execution of the defined test cases on real services.
- *V&V Manager*: includes the policy that underlies the decision of starting a test execution phase or other V&V activities. The V&V Manager will take its decision on the basis of *V&V events* coming from the run-time environment and distributed by an *Event Broker*, possibly implemented using Publish/Subscribe mechanisms.
- *Test*, *Test Suite*, and *Test Case*: describe the structure and the composition of test cases to be executed to evaluate a service. Due to *scalability issue*, most of the elements in the structural models are available in more than one instance at run-time. As a consequence, the different elements will be dispersed within a CHOReOS infrastructure and will be able to act independently.

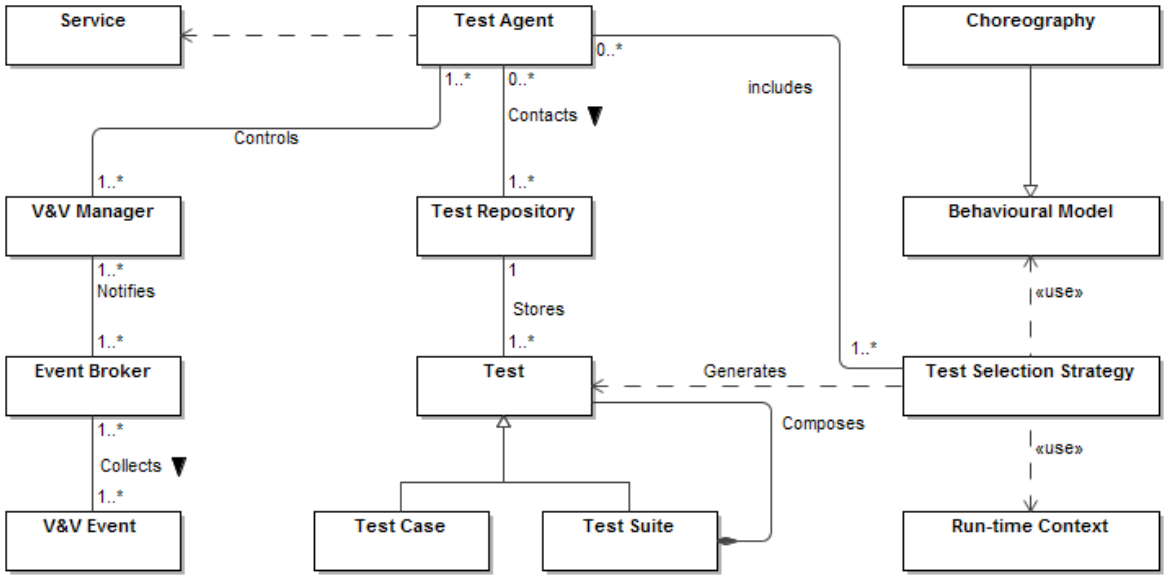


Figure 3.13: Structure View of the Testing Activities (Analysis Concern)

3.3.2. Data Flow

In the following, we introduce and discuss the data flow diagrams for the dependency analysis, QoS prediction, and testing activities.

Dependency Analysis

We produced a data flow diagram for each of the use cases presented in Figure 3.11. We focus the discussion on the kind of data that is generated and exchanged among tasks in the data flow diagrams:

- *Perform Participant Change Impact Analysis*: This is an intra-choreography analysis (Figure 3.14) that starts when the actor informs the choreography participant for which he wants to conduct a change impact analysis (P1 in the example). After that, the dependency analyzer computes the participants' call-graph by inspecting the message exchanges that occur in the subject choreography. Once this is accomplished, a series of SNA metrics and algorithms are calculated to determine the change impact of P1. As a result, a report is generated and sent back to the actor.

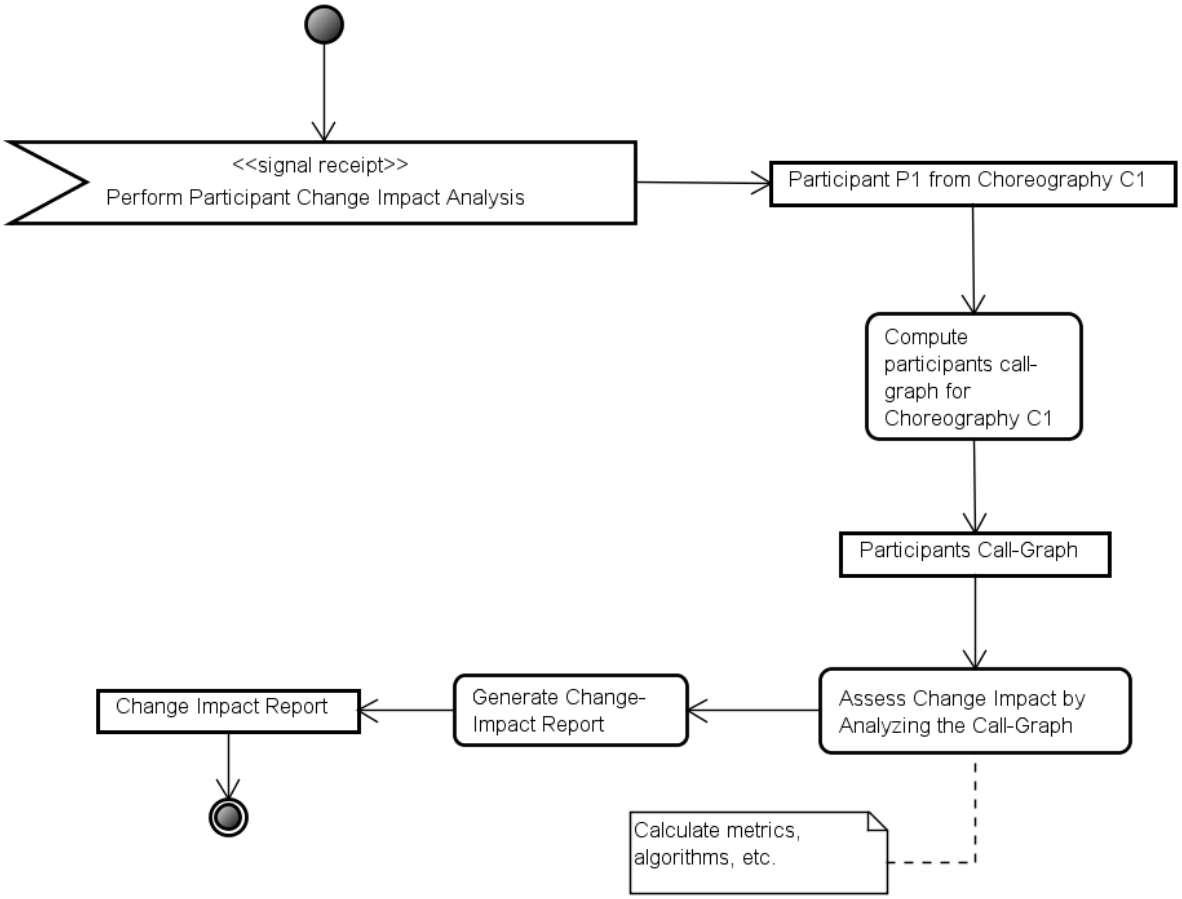


Figure 3.14: Perform Participant Change Impact Analysis functionality

- *Perform Choreography Change Impact Analysis*: This is an inter-choreography analysis (Figure 3.15) that starts when the actor informs the choreography for which he wants to conduct a change impact analysis (C1 in the example). After that, the dependency analyzer asks the Choreography Registry for the enacted choreographies and then computes the choreographies' call-graph. Once this is accomplished, a series of SNA metrics and algorithms are calculated to determine the change impact of C1. In particular, the likelihood of impact is determined by analyzing the schema of each possibly impacted choreography [12]. As a result, a report is generated and sent back to the actor.

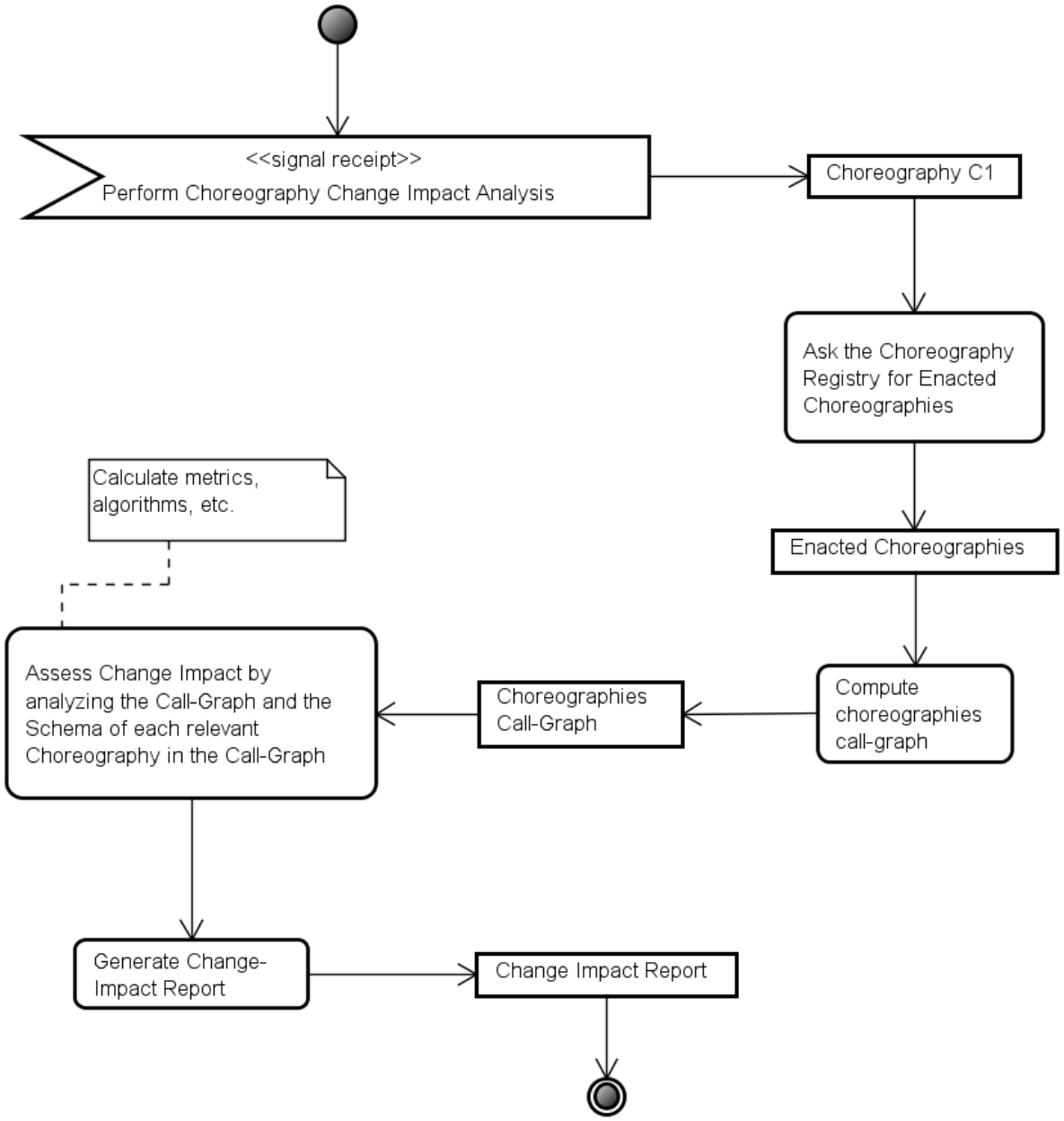


Figure 3.15: Perform Choreography Change Impact Analysis functionality

- *Perform Service Change Impact Analysis*: This post-service-binding analysis (Figure 3.16) that starts when the actor informs the service for which he wants to conduct a change impact analysis (S1 in the example). After that, the dependency analyzer asks the Choreography Registry for the enacted choreographies and then two activities are performed in parallel. One consists in discovering which participants are impacted by a change in S1 (left-hand side of the diagram) and determining whether a service substitution is feasible for such services. This is accomplished with the help of the Discovery Registry and possibly other CHOReOS components. The other activity consisting in calculating the choreographies' call-graph (right-hand side of the diagram). Once these two main activities are finished, the service change impact is finally evaluated (taking into account the preliminary report, the call-graph, and the schema of relevant choreographies) [12]. As a result, a report is generated and sent back to the actor.

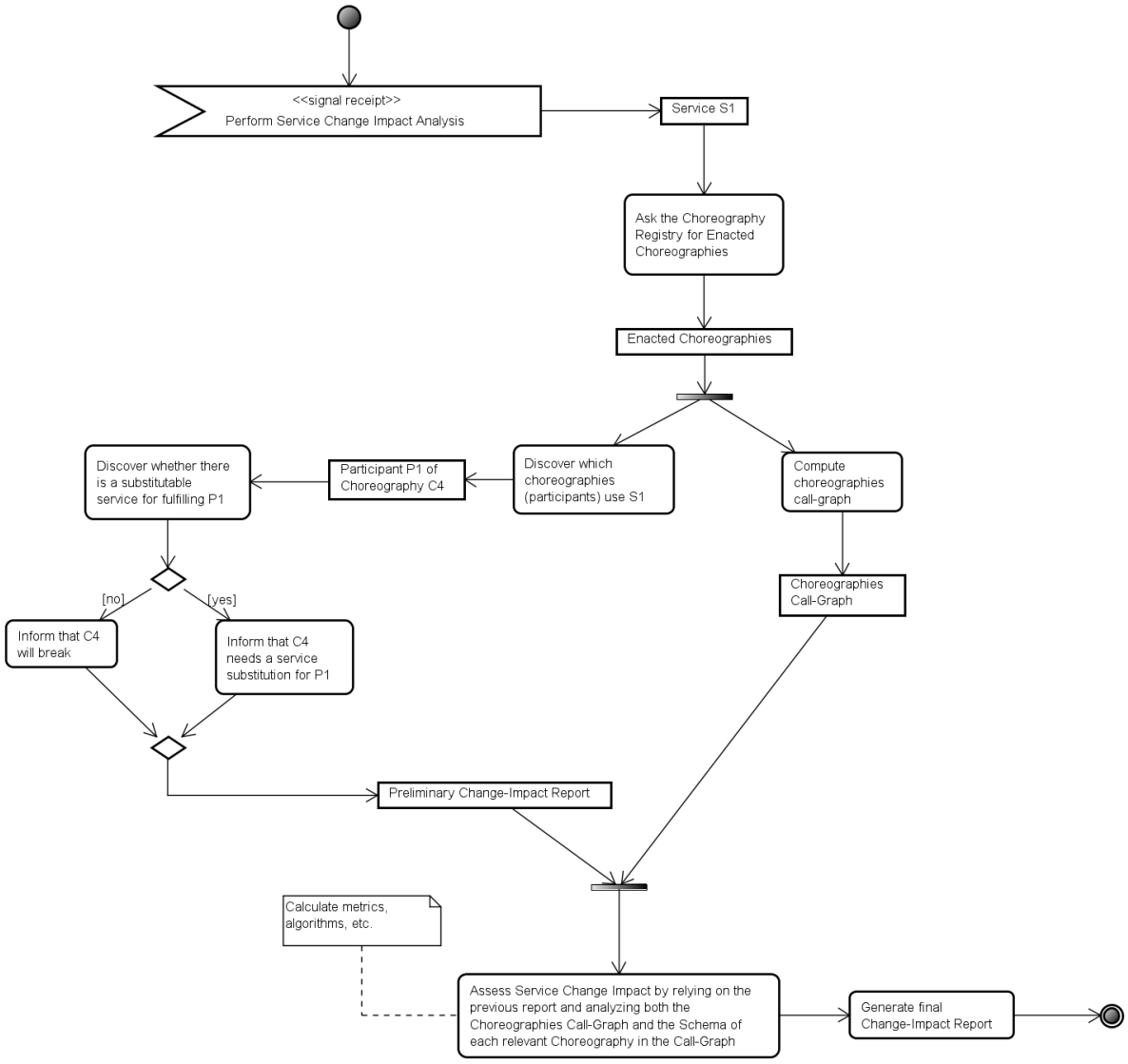


Figure 3.16: Perform Service Change Impact Analysis

QoS Prediction

The dataflow for QoS Prediction is depicted in Figure 3.17. Given a choreography for which QoS, the Predictor first interacts with the discovery registry, the choreography registry and the choreography model in order to collect structural and non-functional information. The objective of this stages is to capture the choreography as a three level graphs (Hierarchical Service Graph [5, 6]) capturing interactions between services operations, services that are used and their hardware deployment. Once obtained, a parsing algorithm is used to generate from the HSG a set of constraints that captures the reputation, availability, price, response time, energy Consumption of the captured HSG. The run of this linear program returns for the choreography the QoS value that can be expected in each dimension.

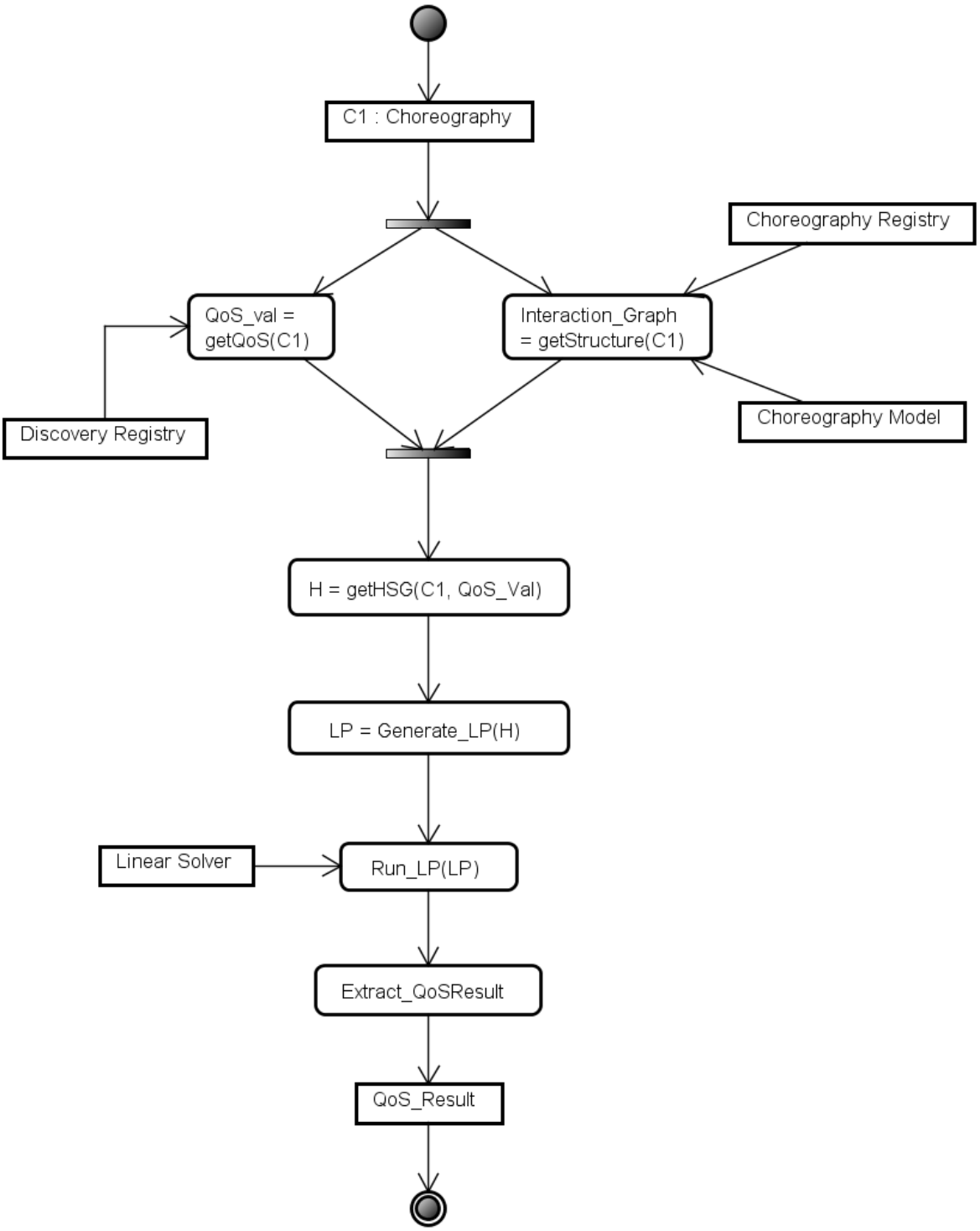


Figure 3.17: Predict Choreography QoS Functionality

Testing Activities

The data flow of the new *Test Activation* functionality is shown in Figure 2.8(b) at pag. 12. Essentially, the run-time testing system will be mainly activated at run-time in order to execute a testing session to evaluate a service. The activation is triggered by a specific event whose occurrence leads to verification activities. Such event could be part of management activities. After the activation, if an appropriate `Test suite` is available in the `Test Repository`, then the `Test Case Selection`

activity is conducted and a `Test is produced`. If the test suite is not available it is derived according to the choreography specification and added in the `Test Repository`. Finally, generated tests are executed against the service under test. At this stage, we do not explore the possibility of performing specific activities in the case where executed test cases have highlighted behaviors that are not in-line with those specified in the considered choreography.

3.4. Messaging Concern

The *Messaging Concern* of the CHOReOS conceptual model addresses in a generic way the communication capability that allows services to interact with each other in the Future Internet. In the elicitation of the CHOReOS Messaging Concern, we strongly rely on the *Messaging Concern* of the NEXOF-RA conceptual model, which we extend in several ways. To point out this reliance, we maintain the term *Messaging Concern*, although, in CHOReOS, interaction may be performed through the exchange of messages, events, or data, as supported by the service communication middleware platforms. Nevertheless, we extend the notion of *message* to represent all these different ways of interaction. More specifically, in CHOReOS, we account for the heterogeneous interaction paradigms employed in the various middleware implementations, both in our elicitation of the CHOReOS architectural style (see Deliverable D1.4b [18]) and its realization in the CHOReOS middleware (see Deliverable D3.1 [14] and subsequent WP3 deliverables). In particular, we provide comprehensive abstractions as well as conversion mechanisms for interoperability among different interaction paradigms, and we implement those in the eXtensible Service Bus (XSB). Hence, the CHOReOS *Messaging Concern* should represent conceptually both the interaction diversity inside the Future Internet and the CHOReOS interoperability solution.

The functionalities of the CHOReOS *Messaging Concern* are depicted in the use case diagram of Figure 3.18. Our extended *Exchange of Message* functionality denotes an *end-to-end exchange of data (including control)* between *Communication Parties*, where the semantics of this exchange may take different forms, such as synchronous or asynchronous, one-to-one or one-to-many, via a direct message or a brokered event or a write/read operation in a shared memory. The *Message Broker* entity represents non-direct interaction. As further depicted in the use case diagram, the *Exchange of Message* functionality includes the *Validation of Message* and *Routing of Message* functionalities, commonly supported by the communication middleware. In the following sections, we give more details by introducing the structure view and the data flow view of the message concern.

3.4.1. Structure View

Figure 3.19 depicts the information model of the message concern. The core concept is (our extended) *Message*, basic unit of communication between parties. A message is characterized by a *Message*

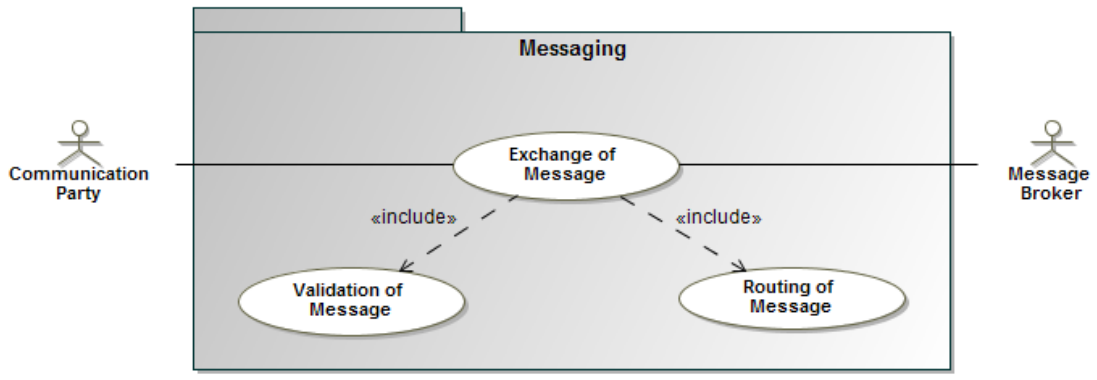


Figure 3.18: Functionalities of the Messaging Concern

Type, classifying the structure and semantics of the message. The *Message Metadata* is associated with the content of the message and is used to process and deliver the message. A message is further associated with an *Endpoint*, an *Interaction Protocol*, and an *Interaction Paradigm*. These concepts specify the entry point of the recipient service that this message should be sent to and also the interaction pattern and protocol that should be followed for correctly interacting with the service. An endpoint may denote a single recipient service, but also a more loosely coupled destination, such as all the communication parties subscribed to a specific topic in a publish/subscribe setting, or all the parties reading a specific tuple from a shared tuple space. The Interaction Paradigm and Interaction Protocol account for the diverse interaction paradigms and associated interaction protocols included in the scope of CHOReOS, such as remote procedure call (RPC), message-based, shared memory, and event-based paradigms.

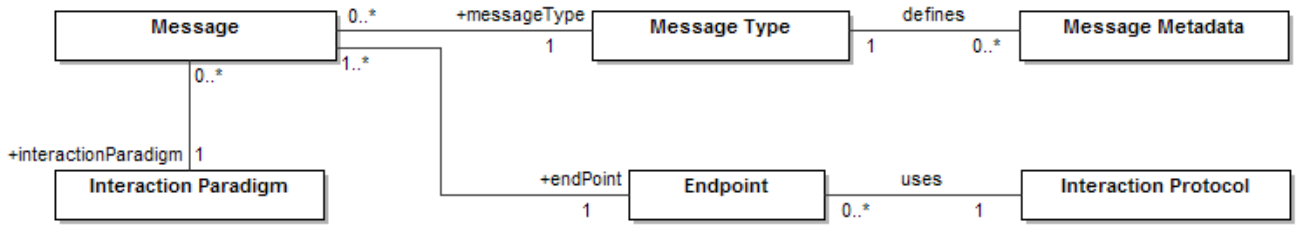


Figure 3.19: Structure View of the Messaging Concern

3.4.2. Data Flow

Strongly relying on the NEXOF-RA message concern specification, we introduce a number of activity diagrams for defining the data flow of the message concern functionalities, as depicted in Figure 3.20, Figure 3.21, Figure 3.22, and Figure 3.23.

Figure 3.20 shows the top-level activity diagram for the message concern, where the functionality *Exchange of Message* processes an arriving message so that it is *dispatched* to the appropriate recipient party, while taking into account the input of *Validation Properties* that serve to validate the message. This (extended) message dispatching employs the Interaction Paradigm and Interaction Protocol discussed above.

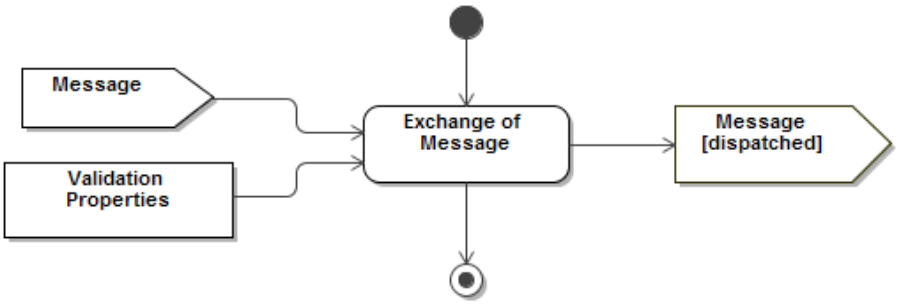


Figure 3.20: Data Flow of the Messaging Concern

Exchange of Message. The functionality *Exchange of Message* is analyzed in Figure 3.21 into the two sub-functionalities already mentioned, i.e., *Validation* and *Routing*. Validation checks the message based on the validation properties (e.g., with respect to its message type or for well-formedness) and validates it or issues a *Rejection Notification* to be delivered to the sender of the message. A validated message is subsequently passed to Routing.

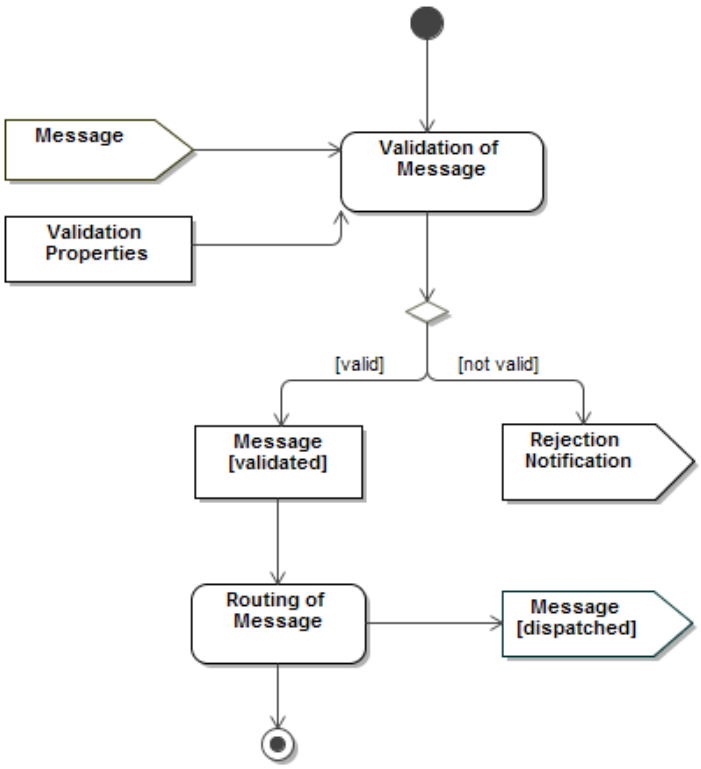


Figure 3.21: Exchange Functionality of the Messaging Concern

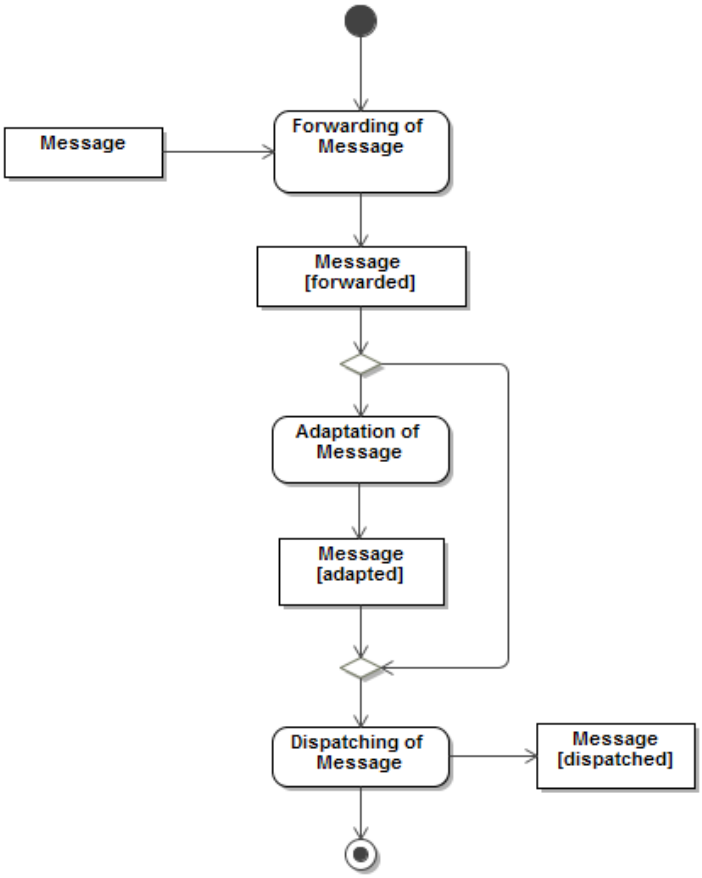


Figure 3.22: Routing Functionality of the Messaging Concern

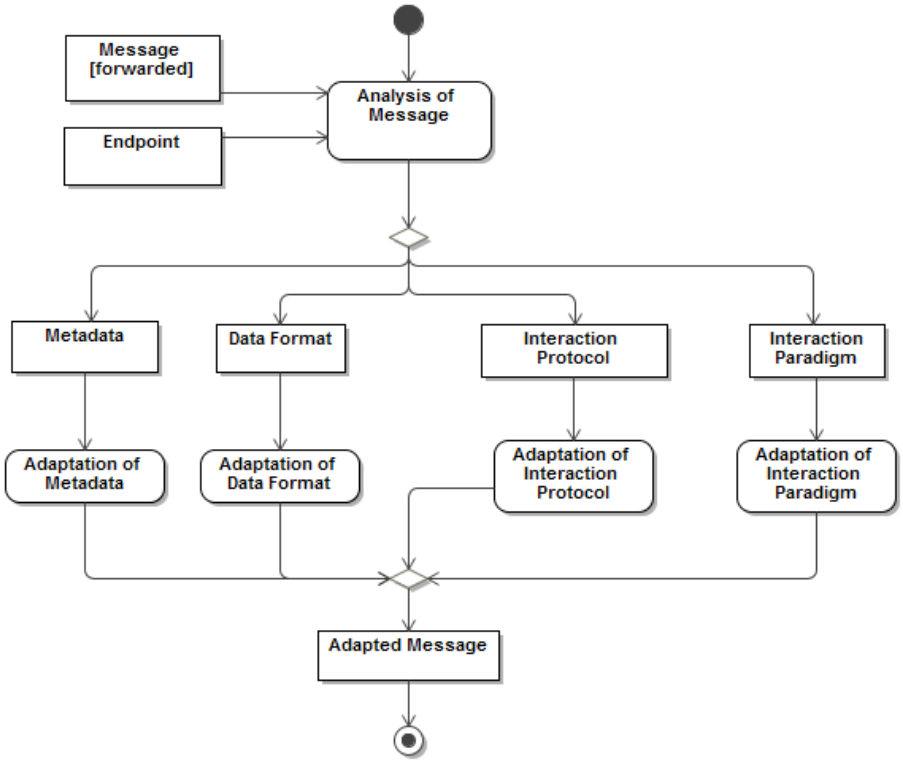


Figure 3.23: Adaptation Functionality of the Messaging Concern

Routing of Message. The functionality *Routing of Message* is analyzed in Figure 3.22. A message is routed to its recipients according to their endpoint information. The message is first *forwarded* to the message broker, which takes care of possibly *adapting* the message and finally dispatching it.

Adaptation of Message. The functionality *Adaptation of Message* is analyzed in Figure 3.23. Based on endpoint (i.e., recipient party) information, message adaptation may comprise different kinds of adaptation, such as *Adaptation of Metadata*, *Adaptation of Data Format*, *Adaptation of Interaction Protocol*, and *Adaptation of Interaction Paradigm* between the sender and recipient parties, when this is required due to related heterogeneity identified between them. We extensively deal with the latter adaptation both in the CHOReOS architectural style, where we elicit abstract connectors representing diverse interaction paradigms and adapt among them to enable their interconnection, and in their realization in the CHOReOS middleware as an XSB, where adaptation takes place via an intermediate bus protocol. Message adaptation, as specified here, should preserve the message semantics. This has been our main objective both in the CHOReOS connectors and in the XSB middleware. Nevertheless, interaction paradigm adaptation can be radical and may introduce some loss or modification of semantics. Hence, we relax this condition, by stating that transformation mappings between interaction paradigms should preserve the semantics as much as possible. In our elicitation of the CHOReOS architectural connectors and of the XSB, we include a precise evaluation of our adaptation solution with respect to the preservation of semantics.

3.5. Service Concern

The service concern is dedicated to capture all the functionalities to support the creation/evolution (specification, design, implementation, and testing), deploying, and execution of services. The *service design and implementation* is not a main concern in the CHOReOS process, since the problem being addressed by CHOReOS is the *composition of thousands of already existing services in the Future*

Internet. However, the NEXOF service concern is extended by the CHOReOS conceptual model to highlight the heterogeneity of services, including things services, and to introduce also the "service instance concept" that is related to business service scalability. Moreover, CHOReOS provides a supporting tool for *software developers*, called *Rehearsal*, to aid in the creation of automated tests during the service creation. The typical activities related to the service concern are pictured by the use case diagram in Fig 3.24. The diagrams of this section were not designed through the refinement process described in the Chapter 2.

In the CHOReOS middleware, the *service deployment* concern is addressed by the Enactment Engine (that will be discussed in Section 3.6), that creates the necessary infrastructure to service execution, such as virtual machines and software dependencies such as operating system packages. The virtual machine configuration and software dependencies of a service are inferred from the service description, that defines the service type (SOAP, REST, etc.), its package type (WAR, JAR, etc.), and its non-functional requirements. The service description contains also properties related to the *composition concern* (see Section 3.1), such as the dependencies among the services within a choreography. One important aspect covered by the service description is the support to external already-running services to participate in the choreography. The Enactment Engine is also structured in two layers: one related to the service concern, that is not aware about choreographies (a composition of services), and another layer that is choreography-aware, which uses the underlying service-concerned layer. The service layer is the responsible for interpreting the service description and preparing the required environment to the service deployment.

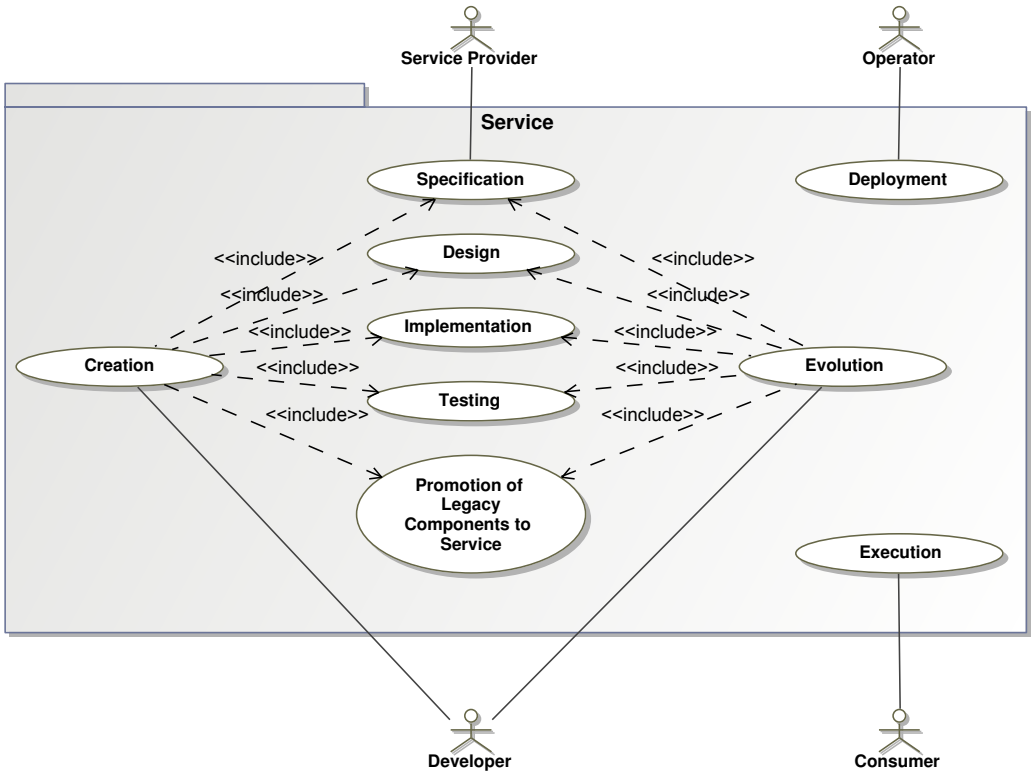


Figure 3.24: Functionality view of the Service Concern.

3.5.1. Structure View

Figure 3.25 presents the service concern structure view, that highlights the heterogeneity of services, including things services. The CHOReOS middleware provides connectors to bind services that use

different communication protocol paradigms, such as client/server, publisher/subscriber, and tuple-spaces. CHOReOS introduces also the “service instance” concept that is related to business service scalability. A service component may be deployed several times with the same configuration, working under a load balancer to provide a single service. Each one of these deployments corresponds to a service instance. Instead using a load balancer, the load distribution may be performed by the services themselves. Another detail introduced by our version of the service concern structural view is the service package, that is the artifact produced from source code and used by operators to deploy the service within a given infrastructure.

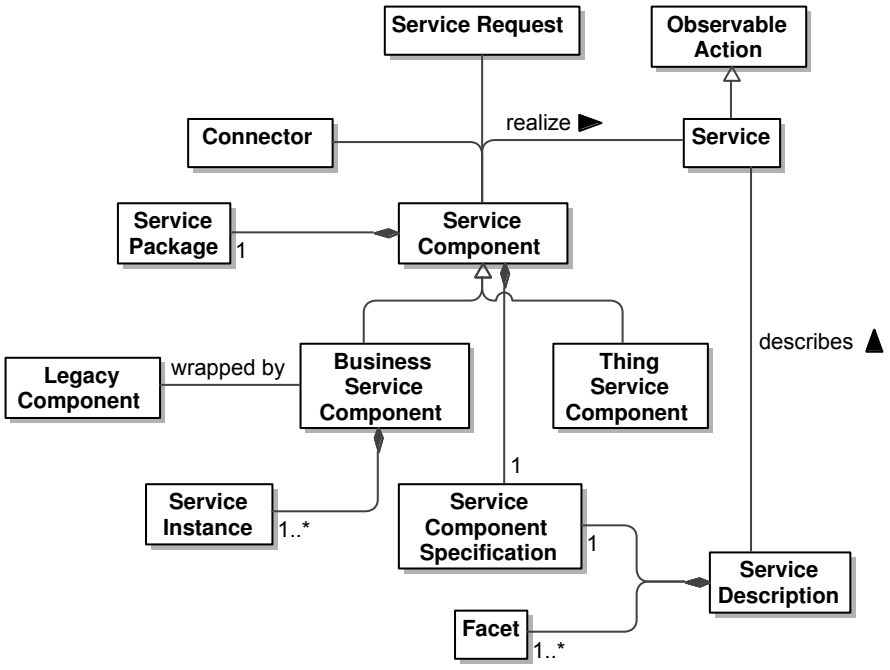


Figure 3.25: Structure view of the Service Concern.

A *service description* represents the information that describes a service. This description may comprise information about several aspects (or *facets* in the NEXOF terminology) of a service. A service facet is a particular information piece that describes a particular aspect of a service, constrained to a particular concern. An important service facet to the service concern is the required information to the deployment process. Such required information includes service type, package type, and non-functional requirements.

The CHOReOS process makes use of different service facets that are expressed in different formats, such as XML and BPMN diagrams. The service facets used in the CHOReOS process include: information required to service deployment, service operations interface (e.g. WSDL), service behavioral interface description, discovery description, information produced by monitoring services.

3.5.2. Data Flow

Service implementations are created to provide some basic functionality for the services that defines a choreography. The result of a service execution is expressed by NEXOF-RA as an observable action. This observable action can either be the change of states of *things* in the real world, but also a message returning some computed data. As shown in the activity diagram in Figure 3.26, the functionality *Creation of Service Component* produces a service package that can be deployed using the Enactment Engine, which results in a running service instance.

A running service instance executes some computation or action in the real world — through a

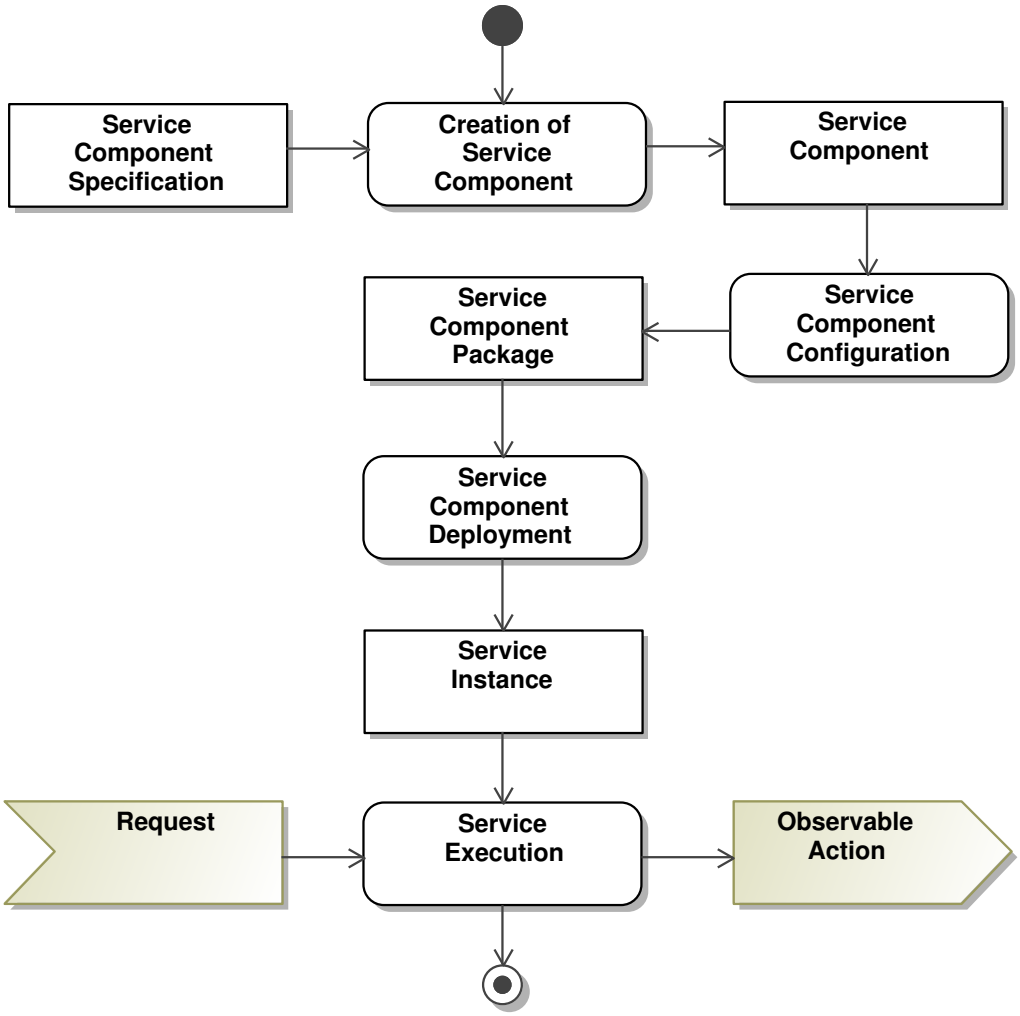


Figure 3.26: Service life-cycle data flow

CHOReOS distributed service bus node (EasyESB) — upon the arrival of a dispatched message requesting the invocation (see the Section 3.4 about the *Messaging Concern*). The result of the execution of the service is an *observable action* satisfying the demand of the service consumer.

The functionality of service execution represents the procedure performed by the service after a client request. After the procedure execution, a response must be transmitted back to the client with the computation result or with the operation status. The execution, i.e. the actual processing of the implemented service logic, is triggered by the invocation, which represents the request of the user asking for the execution of the service. The execution implies in the usage of computational resources that are assigned by the Enactment Engine during the deployment phase. In the next section, we introduce the *Resource Concern*.

3.6. Resource Concern

The *Resource concern* deals with the computational resources needed to enable the execution of all the software components that constitute service platforms, e.g., storage, virtualization of computing and network resources. A basic premise is that the infrastructure is made available as services, regardless of whether they are offered by a cloud computing provider, by a grid platform, or by any sort of computational cluster (as well any combination of them). This Infrastructure as a Service (IaaS) component of

the CHOReOS middleware implements the *virtualized infrastructure service* defined by NEXOF-RA. In very general terms, these are the services that support the execution of choreographies. Typical activities related to the resource concern are pictured by the use case diagram in Fig 3.27. The diagrams of this section were not designed through the refinement process described in the Chapter 2.

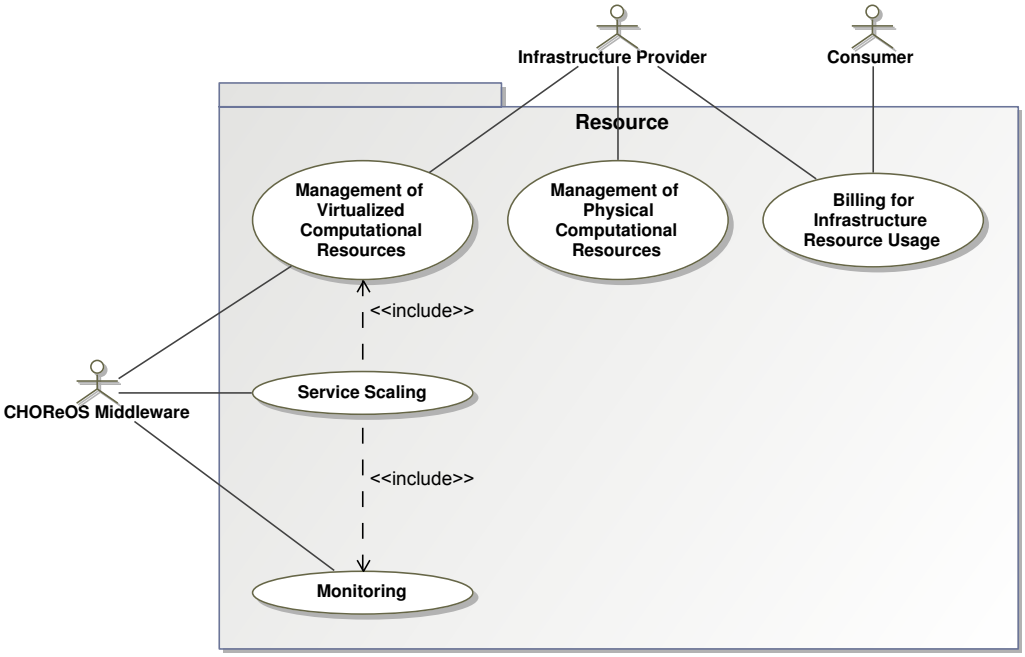


Figure 3.27: Functionalities of the Resource Concern.

The *Enactment Engine* is the actor responsible for managing the deployment/undeployment, starting, and stopping of infrastructure resources. It is the component of the CHOReOS middleware responsible for enabling the enactment of choreographies, creating the virtualized underlying infrastructure, and deploying the services on the target platform. Based on information gathered from monitoring, Enactment Engine will also handle the computational resources to make services scale up or scale down depending on current demand.

Note that although the life-cycle management of virtualized resources is controlled by the Enactment Engine, creating a virtual machine is actually performed by the infrastructure provider, whose services are invoked by the Enactment Engine. Deciding the physical machine where to place a virtual machine, as well all the physical infrastructure management, is the service provider's responsibility. The infrastructure provider will also bill the service provider for the used resources.

3.6.1. Structure View

The *Enactment Engine* receives the formal request for deploying a choreography within an infrastructure provider. As depicted in Figure 3.28, in such request, it receives *service component* descriptions, characterizing the choreography services that are going to be deployed and executed on the target platform. A *service component* description includes all the information needed to instantiate the described service, as well as the information needed for automatically managing the service lifecycle. The *Enactment Engine* also provides an interface to service lifecycle management over a federated virtualized infrastructure.

QoS guarantees can be enforced by the CHOReOS middleware using the standardized interface provided by the *Enactment Engine* component to service replication. Other CHOReOS components, such as monitoring, are able to apply *elasticity rules*, dynamically changing the allocation of the resources

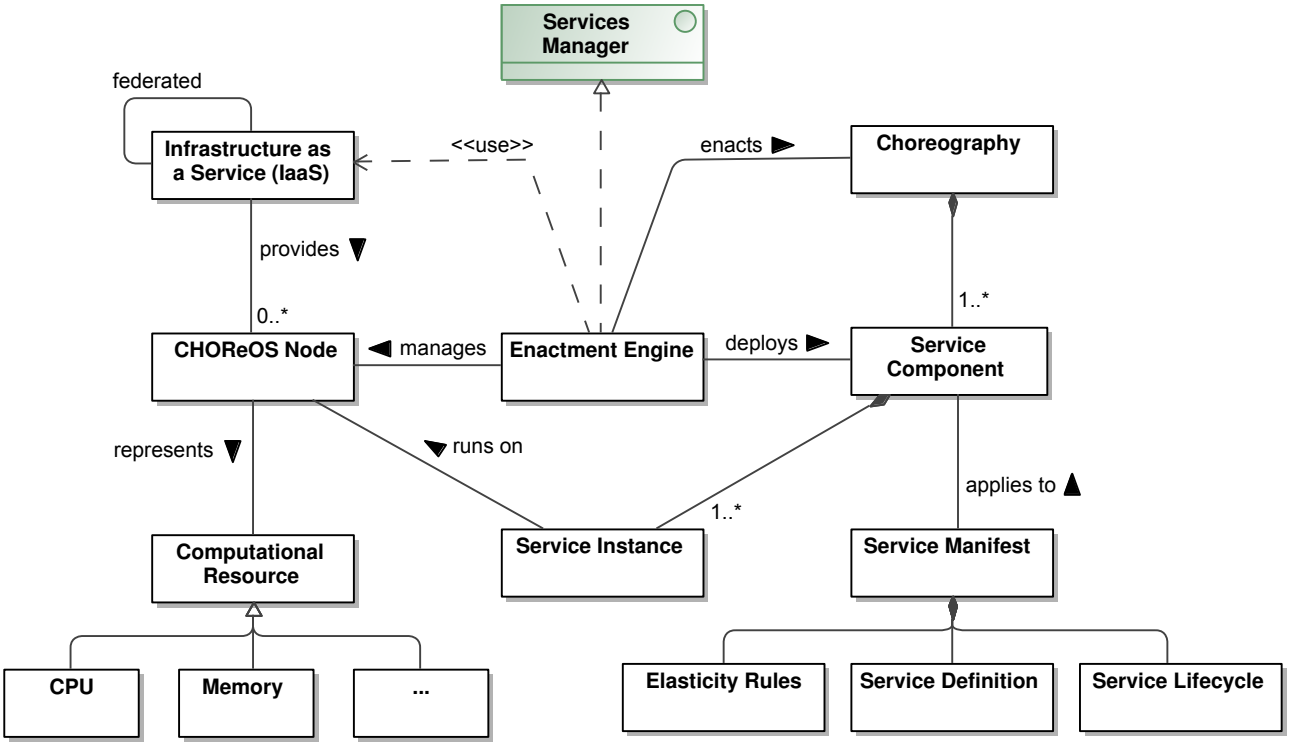


Figure 3.28: Resource Concern structure view.

provisioned by infrastructure services. The underlying computational resources, created by the virtualized infrastructure service, are exposed as virtual machines. The virtual machines, also called *CHOReOS nodes*, are allocated and managed by the Enactment Engine. *CHOReOS nodes* are the basic unit of resource management, abstracting away the physical characteristics of the *infrastructure resources* and enabling physical resource sharing. A *CHOReOS node* is properly configured by the *Enactment Engine* with an operational system and supporting applications (e.g. Java EE containers) to enable the execution of services.

The CHOReOS middleware delegates to the infrastructure providers responsibilities related to the management and execution of the actual virtual machines on the physical infrastructure. An infrastructure provider is free to find the best placement for the virtual machines among the available physical machines that is best suited for its needs. However, the *Enactment Engine* provides a high level interface to manage *CHOReOS nodes*, abstracting differences among different infrastructure providers (cloud, grid, etc.). This common abstraction also enables the creation of federated infrastructures composed of multiple infrastructure providers.

3.6.2. Data Flow

The synthesis process, or a script developed by service operators, sends a choreography specification to the Enactment Engine. This specification contains the non-functional requirements to services execution. Such non-functional requirements may be mapped by the Enactment Engine to virtual machines characteristics that support the requirements, such as the amount of storage or the RAM size. This selection is an enactment engine extension point and can be customized by the enactment engine operator. After deployment, the enactment engine client receives a response with information about the deployed services, such as their URIs and the addresses of nodes for each deployed service. The data flow of this deployment activity is pictured in the Figure 3.29.

The CHOReOS nodes are instrumented in order to monitor their usage. As seen in Figure 3.29,

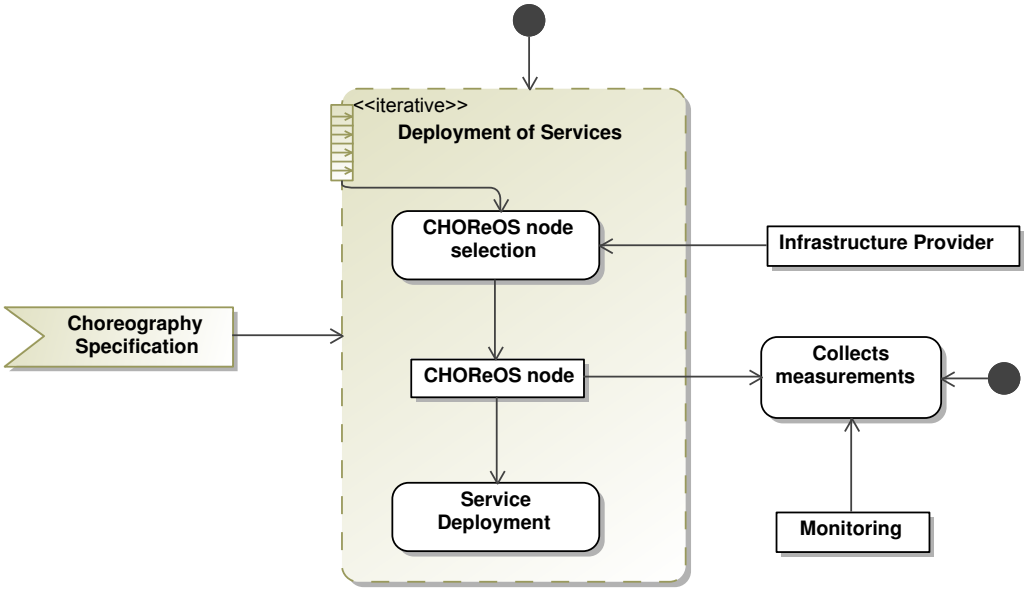


Figure 3.29: Resource Concern data flow.

the CHOReOS monitoring components may collect measurements that are used to support dynamic reconfiguration decisions, and may be used to estimate not only the performance metrics, but also information that could be used to foresee infrastructure billing costs (in the case of using a public cloud).

The NEXOF-RA defines the *lifecycle management of infrastructure resource groups* in terms of the configuration and assignment of computational resources. The enactment engine is able to handle these concerns as follows:

Configuration of Infrastructure Resources the enactment engine defines infrastructure configuration requirements (CPU, RAM, storage, etc.) based on services non-functional requirements. It requests to the infrastructure provider the provisioning of the resources with the required configuration.

Assignment of Computational Resources the Enactment Engine is the component responsible for requesting additional computational resources and releasing these resources when they are no longer used. The actual mapping of virtual machines to physical nodes is performed by the infrastructure provider.

3.7. Governance and Management Concern

In the NEXOF-RA conceptual model, a special interest is dedicated to the concern of runtime management. CHOReOS extends this model by providing advanced and choreography-oriented governance capabilities at both design and run times, including testing, discovery, and life cycle management. We consider Governance as being the trilogy of: doing the right things, the right way, for the right stakeholders [10]. The resulting concern includes not only the governance of services and their choreographies, but also the governance of *Service Level Agreements (SLA)* and their *Life Cycles*. Figure 3.30 depicts the functionalities of the *Governance & Management concern*, as addressed in CHOReOS. In this figure we present without delving into the details the main subsystems considered in the governance framework. First, we present the *V&V* Subsystem thanks to which we are able to test services participating in the choreographies thanks to advanced testing strategies. These facilities are related also to the analysis concern presented in Section 3.3. Second, we describe the *Service Registry* Subsystem

as being the ability of discovering a service including publishing it, describing it, and looking for it. This specific functionality is related to the discovery concern presented in Section 3.2. Third, we present the *Monitoring* Subsystem briefly as being the facilities of monitoring at runtime services, events, choreographies, etc. Finally, the governance and V&V framework provides a very useful functionality that enables the management of service level agreements. In more details, the functionalities embedded in the Governance & Management concern are subdivided into:

- *Governance*, which further includes:
  - *Service Governance* that provides functionalities around services life cycle, publication and discovery, as well as assigning service level agreements. This functionality is provided thanks to the prototype EasierGov that is implemented within WP4. It provides the needed APIs for handling the basic components (Service, SLA, Event, etc.). Moreover, the Service Governance is related to the discovery concern. The Governance Framework is implemented in order to be plugged into the eXtensible Service Discovery middleware in order to populate the Abstraction oriented service base management with business services.
  - *SLA Governance* that encompasses the *SLA Management*, and the *SLA Registry* functionalities. The SLA Management functionality is responsible for providing abilities for *creating*, *updating*, *negotiating*, and *removing* SLAs. The SLA Registry functionality enables the *discovery*, *publication* and *SLA lookup*. A service provider publishes services and SLA templates including the services behavior according to runtime parameters. The service consumers access the SLA registry and select the required SLA templates according to their needs. Once selected, the SLA templates are negotiated between service providers and consumers.
  - *Choreography Governance* that implements the *Choreography Registry* and the *Choreography Enforcement* functionalities. The *Choreography Registry* allows the publication, discovery, and lookup of choreography models and samples. The Choreography Governance and discovery at design time is provided thanks to ServicePot which is implemented in WP4 work package. Services involved in the choreography are assessed at runtime as well as their Service Level Agreements.
- *Runtime Governance* that deals mostly with runtime concerns. It defines a *runtime management* functionality that is itself divided into the *monitoring* and the *test activation* functionalities. Within CHOReOS, we implement a monitoring framework composed of three layers enabling the monitoring of services, service level agreements, and the hardware cloud resources. The CHOReOS Monitoring Framework is event-based as events generated from the middleware layer where the services are deployed as well as from the cloud layer are directed to the Complex Event Processor that assesses the rules are respected. Furthermore, the Test activation concern enables the verification of the SLAs constraints and policy rules at runtime, and relates to the Analysis concern presented in Section 3.3.

3.7.1. Structure View

We extend the NEXOF reference diagram with classes expressing services, choreographies and SLAs, their life cycle, and the governance concerns. Service Level Agreement management has been addressed in the NEXOF Project as the management of the contracts between service providers and consumers. This functionality is also required in CHOReOS in order to allow the negotiations between services consumers and services providers. This is extensively addressed in WP4 on Governance and V&V. The NEXOF Structure view is entirely revised to include functionalities needed by CHOReOS. Moreover, a modeling effort has been provided in order to generalize the structure view used in NEXOF and extend it to cover the Choreography concern.

The diagram in Figure 3.31 presents the structure view of the Governance and Management concern of the CHOReOS initial reference model. The view includes:

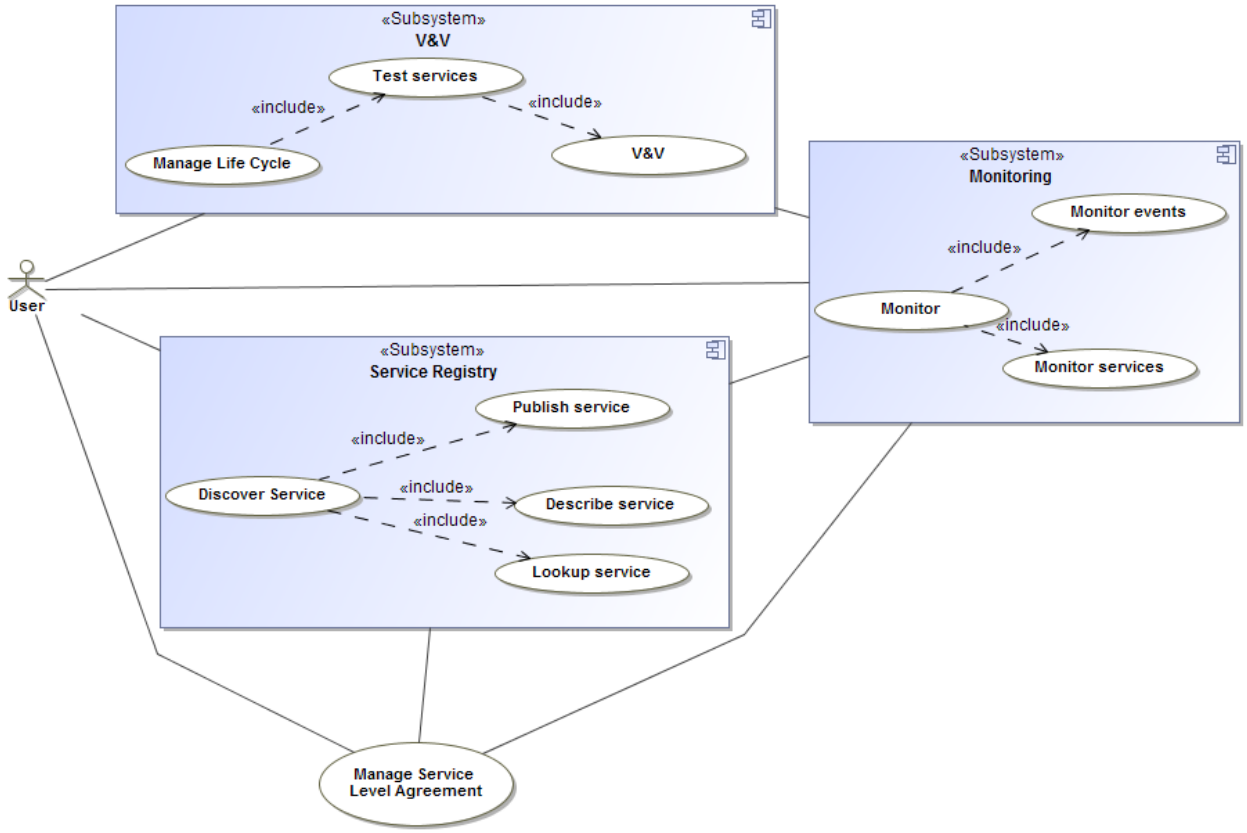


Figure 3.30: Functionalities of the Governance Concern

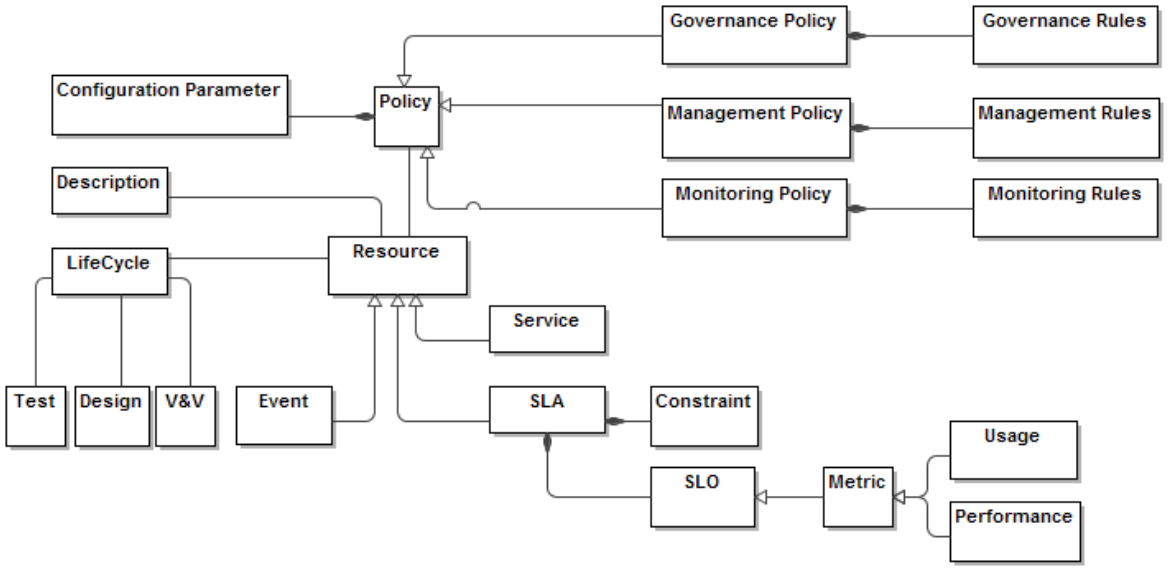


Figure 3.31: Structure View of the Governance Concern

- *Resource* where we consider the following elements: *services*, *events*, and *SLAs*. SLAs are composed of several *SLO* (Service Level Objectives) defining the *performance* and *usage metrics*. An element has a *life cycle* and can be described (*Description*). Finally, *policies* can be applied to elements.

- *LifeCycle* that is associated to each element. The lifecycle covers the elements from design to runtime and helps applying distinctly *governance policies* and *rules* according to each step of the lifecycle. Development, design, and runtime can be part of the life cycle structure. Further structures may be added if needed in order to cover the entire element lifecycle.
- *Policy* that specifies the behavior of an element. *Management*, *governance*, and *monitoring policies* then extend the Policy concept. Each Policy is composed by a set of rules that control the behavior of the elements from design- to run-time. Verification and validation rules are considered as belonging to the governance policy.

3.7.2. Data Flow

In this section the functionalities of the *Governance and Management concern* are presented singularly.

Policy Management. The *policy management* functionality is presented in the data flow diagram in Figure 3.32. *Configuration parameters* are taken into consideration while the *policies* are applied. Policies can concern management, monitoring, and governance. They are later applied to services, events, SLAs. Policies can be *created*, *added*, *updated*, or *removed*.

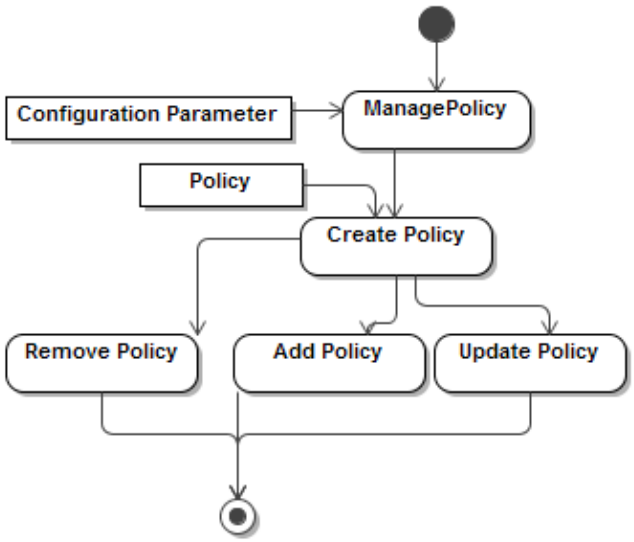


Figure 3.32: Functionalities of the Policy Management

SLA Management. The *Service Level Agreement management* activity enables the *creation*, *update*, *deletion*, and *negotiation* of *SLAs* (see Figure 3.33). SLAs templates are created by the service provider. The SLO and *metrics* are instantiated and their values set.

Monitoring. The *monitoring* functionality enables the observation of the services, choreographies, and SLAs in the running environment (see Figure 3.34). *Notifications* can be triggered in order to inform the system about the gathered information. One can set control levels for each resource. The notifications are expressed into *Events*.

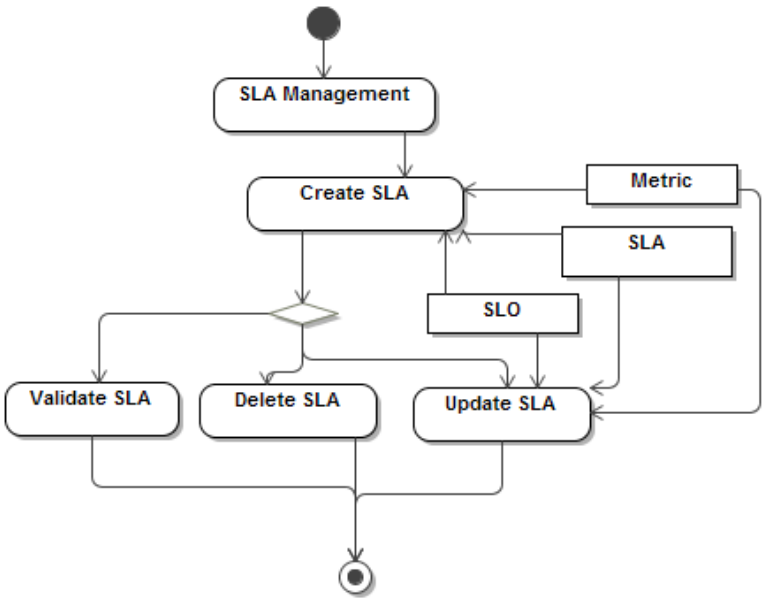


Figure 3.33: Functionalities of the SLA Management

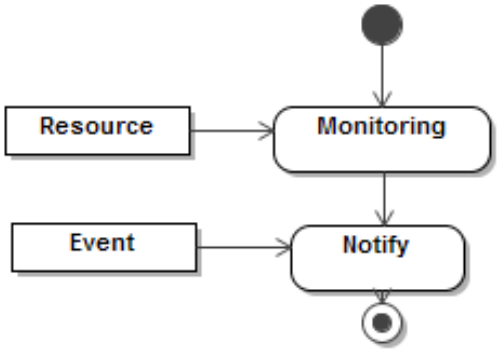


Figure 3.34: Functionalities of the Monitoring

3.8. Presentation Concern

The Presentation concern of NEXOF-RA addresses all mechanisms to enable human users to interact and make use of the functionalities provided by the overall service infrastructure. It aims at supporting developers to work in a highly personalized environment to assemble composite end-user applications from a set of uniform building blocks. The CHOReOS IDRE developed in WP5 (whose requirements are elicited in Deliverable D5.1 [16]) is responsible for realizing the Presentation Concern.

3.8.1. Structure View

Figure 3.35 is an outline of the main tools and related artifacts that developers can use to develop and execute choreographies. In particular, according to the development process described in D2.2 [17] and to the CHOReOS IDRE requirements discussed in D5.1 [16] (and later deliverables of WP5), the development and execution of a service choreography relies on a number of frameworks shown in Figure 3.35.

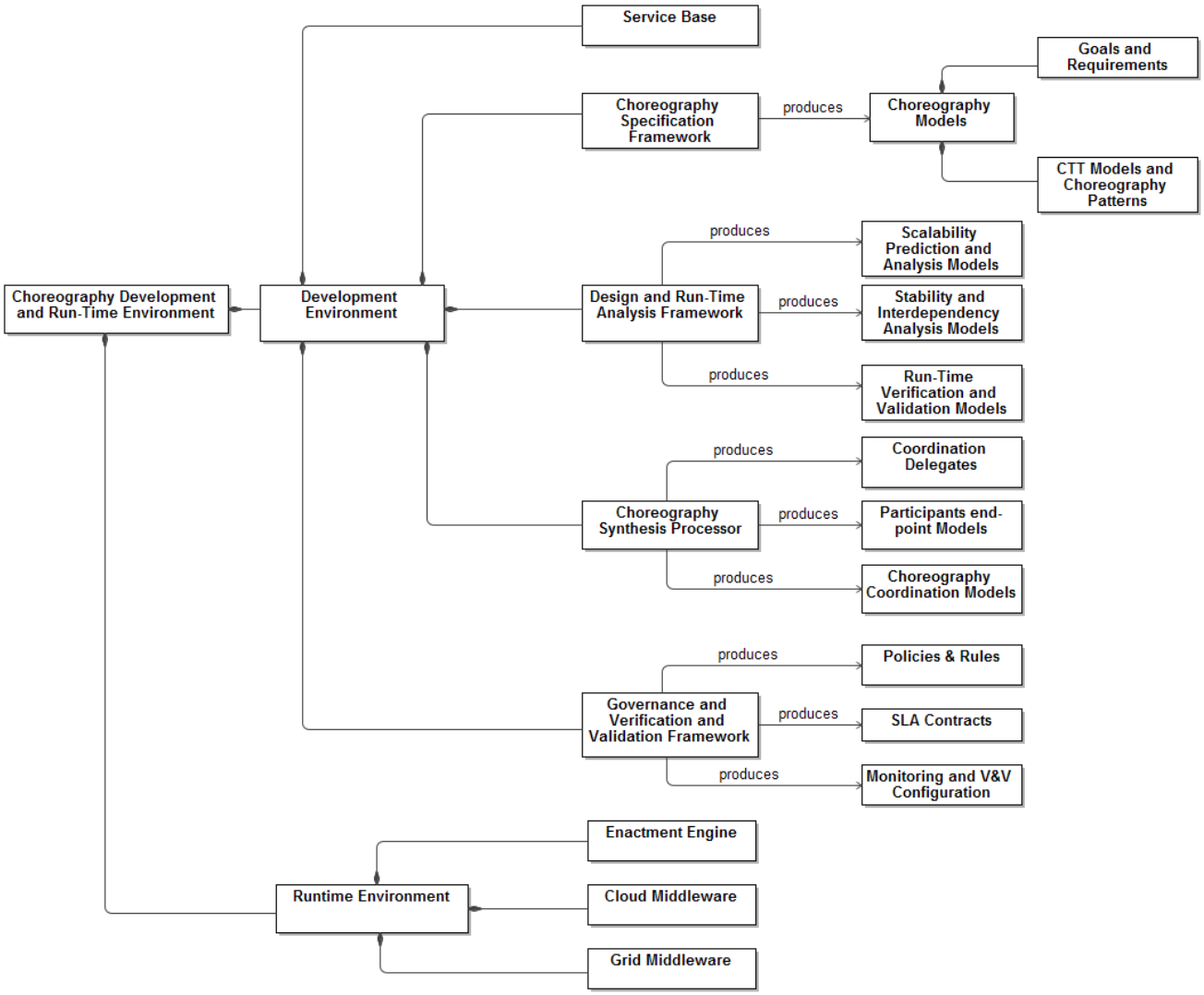


Figure 3.35: Structure View of the Presentation Concern

3.8.2. Data Flow

The data flow related to the Presentation Concern is essentially the CHOReOS development process presented in Deliverable D2.2 [17], which shows all the activities that are executed together with the produced artifacts to develop and execute choreographies. In particular, the CHOReOS *Development Environment* provides sophisticated model-driven techniques and tools that enable domain experts to specify requirements and to model choreographies. After user requirements have been captured via dedicated tools, the choreography is automatically synthesized into a distributed coordination logic. Choreographies are synthesized out of large-scale repositories of services and analysis mechanisms ensure scalability of choreographies.

The synthesized service choreographies are executed on the CHOReOS service-oriented middleware built to sustain stringent requirements in scalability and heterogeneity. This *Runtime Environment* offers runtime capabilities for discovering, deploying, running, adapting, governing, and monitoring ULS choreographies. The CHOReOS middleware provides mechanisms for multi-protocol service discovery, multi-protocol service access, choreography execution, and adaptation for both the Internet of Business Services (IoBS) and the Internet of Things-based Services (IoTS). The Runtime Environment fully leverages the elastic capabilities of *cloud* and *grid* infrastructures.

3.9. Security Concern

The Security concern of NEXOF-RA covers important aspects such as Authentication, Authorization, Privacy, and Integrity. However, while the issue of Security, Privacy, and Trust is acknowledged as a key concern for Future Internet systems, this is not tackled by the CHOReOS consortium, and is left as an area for future work, which may in particular build upon results of relevant European initiatives like the FP7 NESSOS "Network of Excellence on Engineering Secure Future Internet Software Services and Systems" (see http://www.nessos-project.eu/ for more), and the FP7 ANIKETOS "Ensuring Trustworthiness an Security in Service Composition" (see http://www.aniketos.eu for more).

4 Conclusions

This document concludes the work of WP1 for what concerns the definition of the conceptual model. It has reported on the definition of the final CHOReOS conceptual model, that is, a high level common ground capturing the relevant entities/concepts, and relationships among them, underlying the development process and the supporting infrastructures of choreography-based service-oriented systems of the Future Internet.

Since its preliminary version, the CHOReOS conceptual model has been conceived as an extension of the NEXOF Reference Architecture (NEXOF-RA), by introducing new concepts in the NEXOF-RA conceptual model and by refining existing ones. As recalled in this document and detailed in Deliverable D1.2 [15], the NEXOF conceptual model and the NEXOF reference architecture concern general service-oriented systems; whereas, the CHOReOS conceptual model and the CHOReOS IDRE architecture concern the class of service-oriented systems realized as choreographies of services discovered within the Future Internet, and "architected" according to the CHOReOS architectural style.

An important aspect worth to be highlighted is that, since its inception, the conceptual model has been providing an integrated view and a common understanding among the different CHOReOS partners, and has served as high-level input domain model to the development of the RTD work packages WP2 to WP5. At the same time, the conceptual model has been refined/extended according to the outcomes of RTD work packages WP2 to WP5. In this sense, the work undertaken to develop the conceptual model constituted the common ground on which all the CHOReOS partners have built upon to reach a shared view of the methodologies and techniques to be developed and to be integrated in a coherent framework, namely the CHOReOS IDRE.

Bibliography

[1] A. Bertolino, G. De Angelis, A. Di Sandro, and A. Sabetta. Is my model right? let me ask the expert. *Journal of Systems and Software*, 84(7):1089 – 1099, 2011.

[2] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T.J. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2003.

[3] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Modeling Object Language*. Object Technology Series. Addison-Wesley, third edition, September 2003.

[4] Xavier Franch and Gemma Grau. Towards a catalogue of patterns for defining metrics over i* models. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 197–212, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Alfredo Goldman and Yanik Ngoko. On graph reduction for qos prediction of very large web service compositions. In *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*, SCC '12, pages 258–265, Washington, DC, USA, 2012. IEEE Computer Society.

[6] Alfredo Goldman, Yanik Ngoko, and Dejan Milojicic. An analytical approach for predicting qos of web services choreographies. In *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '12, pages 4:1–4:6, New York, NY, USA, 2012. ACM.

[7] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadis, Marco Autili, Marco Aurelio Gerosa, and Amira Ben Hamida. Service-Oriented Middleware for the Future Internet: State of the Art and Research Directions. *Journal of Internet Services and Applications*, 2(1):23–45, May 2011.

[8] Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, November 1994.

[9] N. Maiden, James Lockerbie, Debbie Randall, Sean Jones, and D. Bush. Using satisfaction arguments to enhance i* modelling of an air traffic management system. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 49–52, 2007.

[10] Eric A. Marks. *Service Oriented Architecture Governance for the Services Driven Enterprise*. John Wiley & Sons, Inc. Editions, 2008.

[11] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: an On-line Lexical Database*. *International Journal of Lexicography*, 3(4):235–244, December 1990.

[12] Gustavo Oliva, Marco Gerosa, Dejan Milojicic, and Virginia Smith. A change impact analysis approach for workflow repository management. In *Proceedings of the 2013 IEEE International Conference on Web Services*, ICWS '13 - Application & Experience Track, Washington, DC, USA, 2013. IEEE Computer Society. To appear.

[13] Klaus Pohl. *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, 2010.

[14] CHOReOS Project Team. CHOReOS Middleware Specification - Public Project deliverable D3.1, September 2011.

[15] CHOReOS Project Team. CHOReOS Perspective on the Future Internet and Initial Conceptual Model Final, April 2011.

[16] CHOReOS Project Team. Requirements for the CHOReOS IDRE - Public Project deliverable D5.1, May 2011.

[17] CHOReOS Project Team. Definition of the Dynamic Development Process for Adaptable QoS-aware ULS Choreographies, September 2012.

[18] CHOReOS Project Team. Final CHOReOS Architectural Style and its Relation with the CHOReOS Development Process and IDRE - Public Project deliverable D1.4(b), April 2013.