Deliverable D2.3: CHOReOS Dynamic Development Process: methods and tools

Marco Autili

CHOReOS

Large Scale Choreographies for the Future Internet

ICT IP Project

Deliverable D2.3

CHOReOS Dynamic Development Process: methods and tools

http://www.choreos.eu

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

Deliverable Number	:	D2.3
Title of Deliverable	:	CHOReOS Dynamic Development Process: methods and tools
Nature of Deliverable	:	Report + Prototype
Dissemination level	:	Public
Licence	:	Creative Commons Attribution 3.0 License
Version	:	3.0
Contractual Delivery Date	:	30 September 2013
Actual Delivery Date	:	21 October 2013
Contributing WP	:	WP2
Editor(s)	:	Marco Autili (UDA)
Author(s)	:	Marco Autili (UDA), Paola Inverardi (UDA), Massimo Tivoli (UDA), Amleto Di Salle (UDA), Dionysis Athanasopoulos (UOI), Panos Vasilliadis (UOI), Dimitris Pilios (UOI) Apostolos Zarras (UOI), James Lockerbie (CITY), Gustavo A. Oliva (USP), Alfredo Goldman (USP), Yanik Ngoko (USP),
Reviewer(s)	:	Guglielmo De Angelis (CNR), Sébastien Keller (THALES)

Abstract

This deliverable closes the WP2 work, whose main objective has been the definition of a development process for the development of ULS choreographies. The WP2 work capitalizes on the initial work described in Deliverable D2.1 [28], where we presented a model of the CHOReOS development process by abstractly describing the "strategy" that CHOReOS uses for developing choreographies. Then, by leveraging the abstract model of the CHOReOS development process defined in Deliverable D2.1, in D2.2 [36] we precisely described the specific set of concrete process activities, and related relationships, by using BPMN2 Process Diagrams as graphical notation.

In this deliverable we describe the software that have been developed in WP2 in order to actually realize the final CHOReOS development process defined in D2.2 [36], and refer to the WP3-5 deliverables for the description of the software developed outside WP2. The released tools are made available from the official CHOReOS repository hosted by OW2 and are integrated in the CHOReOS IDRE. For each presented tool we leverage the particular use case(s) whose characteristics best demonstrate the specific functions of the tool.

Keyword List

Dynamic Software Development Process tools, Requirements Specification, Large-scale Service Base Management, Choreography Analysis, Choreography Synthesis, Meta-modeling, Choreography-centric Service Oriented Computing and Architecture, Model-driven Engineering, BPMN2.

Document History

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

Document Reviews

Review	Date	Ver.	Reviewers	Comments
Outline	01 July 2013	1.0	All authors	Preliminary draft version available
Draft	09 September 2013	1.x	Valérie Issarny	Intermediate version released for internal review
QA	16 October 2013	2.0	Guglielmo De Angelis, Sébastien Keller	Editorial comments
PTC	21 October 2013	A	PTC	-

Glossary, acronyms & abbreviations

Item	Description
AoSBM	Abstraction oriented Service Base Management
ATL	Atlas Transformation Language
API	Application Programming Interface
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
BPMN2	Business Process Modeling Notation - ver. 2
CD	Coordination Delegate
CLTS	Choreography LTS - Labeled Transition System
CS	Client Server
DAML-S	DARPA Agent Markup Language for Services
DBMS	Data Base Management System
DSMS	Data Stream Management System
ESB	Enterprise Service Bus
FA	Functional Abstraction
FI	Future Internet
FLTL	Fluent Linear Temporal Logic
FSP	Finite State Processes
GA	Generic Application
HTTP	HyperText Transfer Protocol
IDL	Interface Description Language
IDRE	Integrated Development and Runtime Environment
IOPE	Inputs, Outputs, Preconditions, Effects
IoT	Internet of Things
JBI	Java Business Integration
JSON	JavaScript Object Notation
LTL	Linear Temporal Logic
LTS	Labeled Transition System
LTSA	Labeled Transition System Analyzer
MDE	Model Driven Engineering
MEP	Message Exchange Pattern
MILP	Mixed Integer Linear Programming
NFA	Non Functional Abstraction
OWL	Ontology Web Language
OWL-S	Ontology Web Language for Services
PS	Publish Subscribe
Q4BPMN	Quality for BPMN
RDF	Resource Description Framework
REST	REpresentational State Transfer
RFID	Radio Frequency Identification

RQ	Research Question
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
S & A	Sensor and Actuators
SAWSDL	Semantically Annotated Web Service Description Language
SNA	Social Network Analysis
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-Oriented Computing
SOM	Service Oriented Middleware
SPARQL	SPARQL Protocol and RDF Query Language
STR	STReaming
STR-CS	STReaming Client/Server
STR-PS	STReaming Pub/sub
STR-TS	STReaming Tuple Space
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
WS	Web Service
WSBQL	Web Service Based Query Language
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
WSQM	Web Service Quality Model
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol
XSB	eXtensible Service Bus
XSB	eXtensible Service Discovery

Table Of Contents

List Of Tables

List Of Figures

1 Introduction

This deliverable closes the WP2 work, whose main objective has been the definition of a development process for the development of ULS choreographies. Towards this goal, within WP2, a set of macro activities have been identified and some of the software components implementing these activities have been developed in other WPs, as recalled by the following items:

- Domain expert requirements specification tool (implemented in WP2);
- Large scale abstraction-oriented service base management, registration and query engine (implemented in WP2);
- Choreography synthesis processor and related model transformations (implemented in WP2);
- Choreography deployment and execution, and supporting middleware (implemented in WP3);
- Design and run-time analysis, governance V&V, monitoring and V&V configuration tools (implemented in WP2 and WP4).

In the following we further recall the history of the work undertaken in WP2 by the CHOReOS team:

1) *Deliverable D2.1 [28]* – The WP2 work capitalizes on the initial work described in deliverable D2.1 [28]. In D2.1 we presented a model of the CHOReOS development process by abstractly describing the "strategy" that CHOReOS uses for specifying, analysing, synthesizing and enacting, governing and monitoring ULS choreographies during the whole life cycle (from design to runtime to evolution). The dynamic development process model consists of activities that are common to other development processes, but within CHOReOS they are organized in order to fulfill the specific commitments that have been imposed to deal with ULS service choreographies. In deliverable D2.1 we gave an abstract description that characterizes the CHOReOS development process by defining the main activities that need to be performed and the artifacts manipulated by these activities without referring to specific technologies, tools, standards, models, etc.
2) *Deliverable D2.2 [36]* – In D2.2 [36], by leveraging the abstract model of the CHOReOS development process defined in deliverable D2.1, we precisely described the specific set of concrete process activities, and related relationships, by using BPMN2 Process Diagrams as graphical notation. Moreover, in deliverable D2.2 we specified what standards, notations, languages, technologies, and tools we have then used to implement the tools supporting the defined process activities, and that have been then integrated into the CHOReOS Integrated Development and Run-time Environment (IDRE) developed in WP5.
3) *This deliverable* – In this deliverable we describe only the software that have been developed in WP2, and refer to the WP3-5 deliverables for the description of the software developed outside WP2. To this purpose, we recall that the complete documentation of all the software components herein presented is available at the software documentation pages of the CHOReOS web site[1].

[1] http://choreos.eu/

In this deliverable, for each tool we leverage the particular use case(s) whose characteristics best demonstrate the specific functions of the tool.

The document is structured as follows: the domain expert requirements specification tool is described in Chapter 2. The large scale service base, its management, its registration and query engine are described in Chapter 3. The synthesis processor and the analysis tools are reported in Chapter 4 and Chapter 5, respectively. Conclusions are finally given in Chapter 6.

2 Domain expert requirements specification tool

As reported in deliverable D2.2 [36], we developed a requirements-led method to design service choreographies as part of a more complete framework to design and run more adaptive service-based systems. The method is used by domain experts to specify the requirements that service consumers have on a service-based system, and by choreography designers to adapt their choreography diagrams to satisfy requirements more effectively. The method is supported by the domain expert requirements tool, known as the CHOReOS Requirements Tool.

In this chapter, we provide a brief overview of how the Requirements Tool is applied in the CHOReOS Approach (Section 2.1) with reference to the DynaRoute use case presented in deliverable D8.4 [39]. As the details of the tool are documented on the CHOReOS website[1], we focus on the integrated aspects of the Requirements Tool as part of the wider Choreography Specification Framework. Our latest results on mapping quality requirements to the Q4BPMN approach are presented (Section 2.2), and the chapter finishes with a technical assessment of our tool-supported method applied to the DynaRoute use case (Section 2.3).

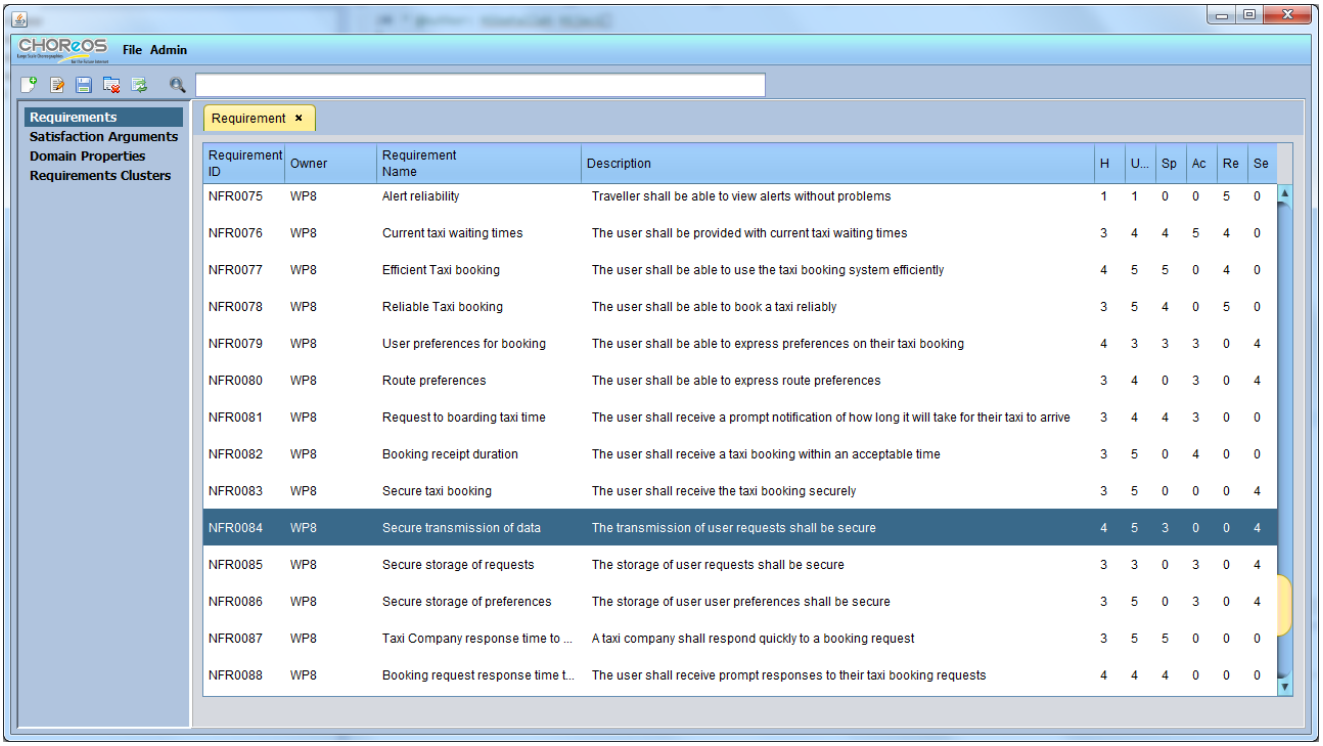
2.1. CHOReOS Requirements Tool at a Glance

The domain expert uses the CHOReOS Requirements Tool to specify requirements on a future service-based software system. To maximize the uptake of the CHOReOS approach, the tool supports the expression of natural language requirements that need domain knowledge rather than analytic training to write. The approach assumes that a domain expert gathers requirements from the consumers of its services and acts as a surrogate for service consumers who are unlikely to participate directly in a requirements process. A further assumption is that domain experts with minimum training are able to express functional requirements in structured natural language effectively (e.g., [15]), but may need help expressing measurable quality requirements such as performance and reliability. To do this, the tool enables the domain expert to add one or more qualifiers [1] indicative of different qualities of the 'root' functional requirement. Figure 2.1 shows an example of domain expert expressed requirements and quality scores from the DynaRoute use case. Further details of this application of tool can be found in deliverable D8.4 [39].

As requirements expressed in the tool by the domain experts are expected to be written on systems rather than service choreographies, the tool provides support to cluster similar requirements that will map onto a single choreography and its elements such as activities and roles. For example, the tool provides the capability to select then cluster requirements based on an automated assessment of the semantic similarity between any pair of requirements in the set of selected requirements (similar to a linguistic approach for large-scale requirements management reported in [19]). Figure 2.2 shows an example of a requirements cluster from the DynaRoute use case. Further details can be found in deliverable D8.4 [39].

Once the cluster has been completed, the domain expert can fire the requirements at a catalogue of reusable user task models to generate a first-cut choreography diagram that satisfies the requirements. A third-party service called TEDDiE applies sophisticated information retrieval techniques to identify and

[1] http://choreos.eu/bin/Documentation/Development_Environment



Requirement ID	Owner	Requirement Name	Description	H	U...	Sp	Ac	Re	Se
NFR0075	WP8	Alert reliability	Traveller shall be able to view alerts without problems	1	1	0	0	5	0
NFR0076	WP8	Current taxi waiting times	The user shall be provided with current taxi waiting times	3	4	4	5	4	0
NFR0077	WP8	Efficient Taxi booking	The user shall be able to use the taxi booking system efficiently	4	5	5	0	4	0
NFR0078	WP8	Reliable Taxi booking	The user shall be able to book a taxi reliably	3	5	4	0	5	0
NFR0079	WP8	User preferences for booking	The user shall be able to express preferences on their taxi booking	4	3	3	3	0	4
NFR0080	WP8	Route preferences	The user shall be able to express route preferences	3	4	0	3	0	4
NFR0081	WP8	Request to boarding taxi time	The user shall receive a prompt notification of how long it will take for their taxi to arrive	3	4	4	3	0	0
NFR0082	WP8	Booking receipt duration	The user shall receive a taxi booking within an acceptable time	3	5	0	4	0	0
NFR0083	WP8	Secure taxi booking	The user shall receive the taxi booking securely	3	5	0	0	0	4
NFR0084	WP8	Secure transmission of data	The transmission of user requests shall be secure	4	5	3	0	0	4
NFR0085	WP8	Secure storage of requests	The storage of user requests shall be secure	3	3	0	3	0	4
NFR0086	WP8	Secure storage of preferences	The storage of user user preferences shall be secure	3	5	0	3	0	4
NFR0087	WP8	Taxi Company response time to ...	A taxi company shall respond quickly to a booking request	3	5	5	0	0	0
NFR0088	WP8	Booking request response time t...	The user shall receive prompt responses to their taxi booking requests	4	4	4	0	0	0

Figure 2.1: CHOReOS Requirements Tool, showing domain expert expressed requirements and qualities scores from the DynaRoute use case

select the models that are the best-fit abstractions of the requirements problem[2]. The Requirements Tool then automatically reasons with the selected models to generate a first-cut model of an under-constrained choreography diagram that can satisfy the requirements. Figure 2.3 shows an example of a first-cut choreography diagram from the DynaRoute use case. A description of the model can be found in deliverable D8.4 [39].

The final output from the Requirements Tool is an XML file containing elements and relationships of an incomplete BPMN choreography diagram, as derived from the retrieved CTT models, along with the requirements and their relationships to the choreography tasks. The choreography designer is able to import this XML file into MagicDraw, the UML/BPMN compliant modelling tool used in the CHOReOS project. The designer can then use this model to complete the design of the choreography using the derived requirements associated to each choreography activity. Figure 2.4 shows a matrix mapping requirements from the DynaRoute use case to the choreography diagram tasks. A description of this can be found in deliverable D8.4 [39]. Finally, quality requirements on both the choreography and each of its activities are specified using the Q4BPMN extension of BPMN (see Section 2.2).

As can be seen from this DynaRoute example, a successful application of the approach transforms functional and quality requirements on a system expressed in natural language into a single service choreography specification that is expressed using BPMN and Q4BPMN. The choreography is optimized to satisfy the requirements and the domain constraints extracted from matched user task models. Beyond this, the specification of each choreography activity can then be transformed into the specification of requirements monitors on each activity in the choreography specification, as described in both deliverables D4.3 [42] and D4.4 [44].

To put the CHOReOS Requirements Tool into context, it is part of the Choreography Specification Framework, which provides tools to transform requirements expressed on service-based systems to QoS-aware choreography specifications. Below, we provide definitions of the components, as shown

[2]The work builds on a library of domain-independent models developed in S-Cube, the EU-funded Network of Excellence for Software Services

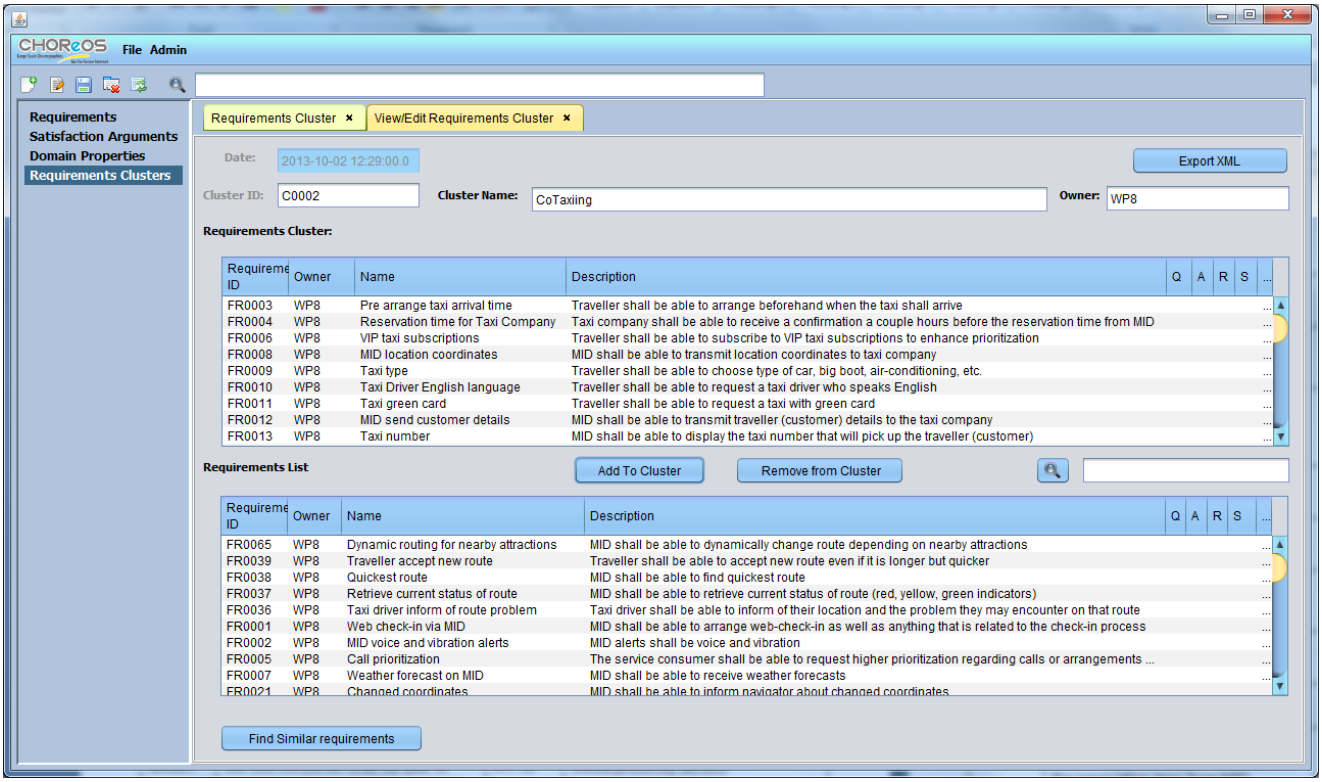


Figure 2.2: CHOReOS Requirements Tool, showing a requirements cluster from the DynaRoute use case

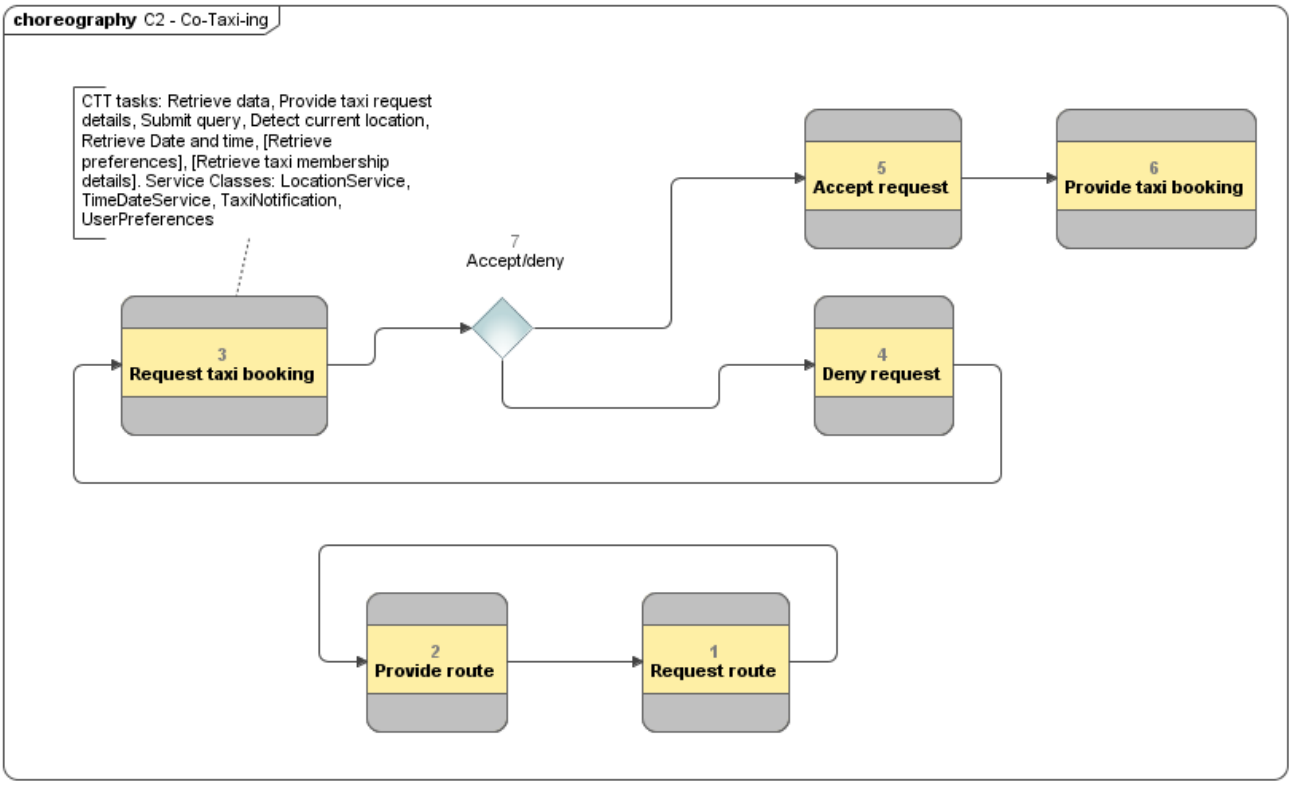


Figure 2.3: MagicDraw, showing a first-cut choreography diagram imported from the CHOReOS Requirements Tool, from the DynaRoute use case

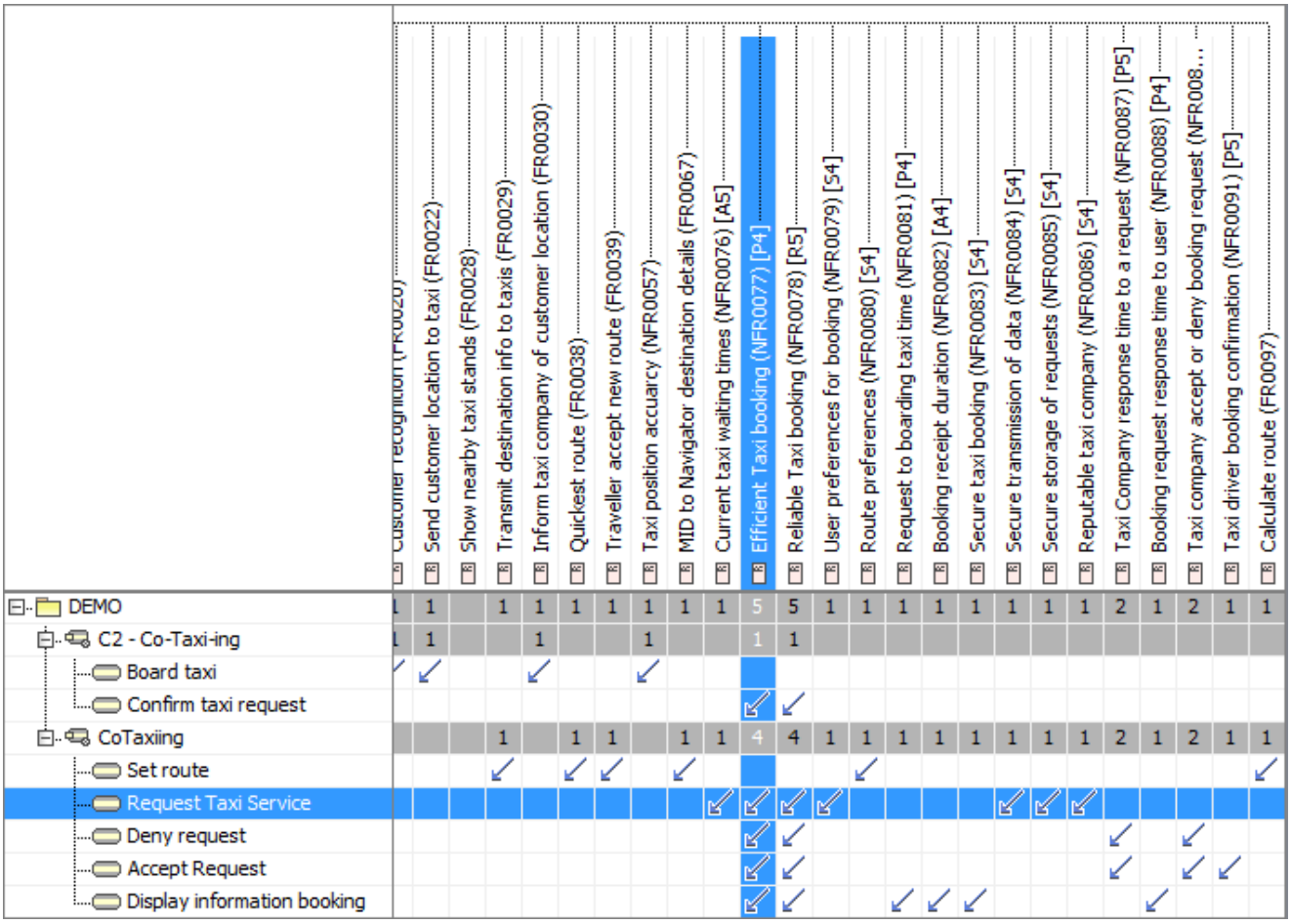


Figure 2.4: MagicDraw, showing a completed requirements-choreography task matrix from the DynaRoute use case

in Figure 2.5, and a description of how they interoperate to support the requirements-led approach described above.

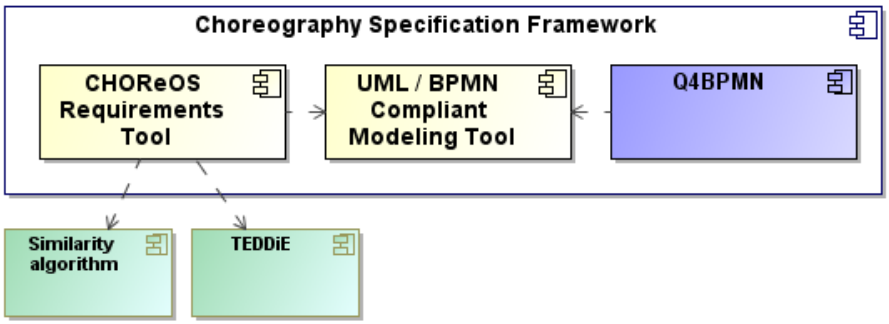


Figure 2.5: CHOReOS Specification Framework components

CHOReOS Requirements Tool: a java-based application that provides front-end access to a local requirements database (MySQL), coupled with requirements analysis functions and access to third party services. The primary purpose of the tool is to provide support for transforming a large number of functional and quality requirements on a system to a single choreography specification. It provides bespoke functionalities including: guidance for expressing natural language requirements on a service-based

software system; functions to identify and cluster relevant requirements for a single choreography; and a service to match the cluster of requirements to a catalogue of user task models which inform the design of a first-cut service choreography.

Similarity algorithm: a third-party service invoked by the Requirements Tool to compute a measure of similarity between any pair of natural language requirements. Full details of this component can be found in deliverable D2.1 [28].

TEDDiE: the TEDDiE algorithm, developed for the EU funded S-CUBE project (http://www.s-cube-network.eu/), was reconfigured as a service for CHOReOS and tailored for use in choreography design. The requirements tool invokes the service to retrieve user task models to provide the basis for a first-cut BPMN Choreography Diagram. Full details of the application of this service can be found in deliverable D2.2 [36].

MagicDraw: the UML/BPMN compliant modelling tool that receives and displays the final output from the Requirements Tool. The XML file output includes: choreography tasks derived from the CTT models and documented with the underlying user tasks and associated service classes; choreography task relationships; the requirements from the cluster; and trace relationships between those requirements and the choreography tasks. Details of the integration between the Requirements Tool and MagicDraw are provided in Section 2.3.

Q4BPMN: an approach for directly annotating BPMN Choreography Diagrams with quality requirements the services entering the choreography will have to abide by. A mapping between domain expert expressed requirements and Q4BPMN has been developed for CHOReOS, and is described below in Section 2.2.

2.2. CHOReOS Requirements Tool Revised

Our main focus for extending the tool support was to better address the quality requirements-driven aspect of choreography specification. Specifically, our work looked at mapping quality requirements from the Requirements Tool with non-functional properties expressed on the choreography diagram using Q4BPMN.

2.2.1. Expressing QOS-aware choreographies

The expression of a QoS-aware choreography begins in the Requirements Tool, where the domain expert is supported to add qualifiers [1] indicative of different qualities of the systems requirements they are writing. The qualities the CHOReOS project considered important for service choreographies were: accuracy, reliability, performance and security. These qualities relate to the CHOReOS Quality Model, presented in deliverable D2.1 [28]. The Requirements Tool enables the user to indicate the importance of each quality on a Likert scale of 1 (very low) to 5 (very high), with the tool ensuring that only be one main quality can be specified.

As reported in deliverable D2.2 [36], the systems requirements are matched to context-rich user task models to automatically generate a candidate BPMN choreography diagram. This model provides the framework with which to structure the requirements and to express qualities on the service choreography. To do this, we use Q4BPMN (Quality for BPMN), a semi-formal notation for specifying quality annotations at the choreography level [6]. It allows choreography designers to extend BPMN choreography diagrams with quality properties without the need for additional models.

2.2.2. Mapping requirements to Q4BPMN

Each response on the scale for each quality type has been associated with a predefined range of measures using Q4BPMN to determine satisfaction with the requirement. Figure 2.6, shows an instantiation of the model applied to the DynaRoute Use case. The four main qualities that can be associated with the user expressed systems requirements correspond to four user quality goals on the wider service-based system - accuracy, dependability, performance and security.

Given the broad scope of these quality goals, we looked at how these qualities could be refined from an agent-based perspective. We identified 3 agent-based quality goals, or 'concerns' as reported in deliverable D4.4 [44]. The concerns were defined as: (i) specifications related to the software system, e.g. response time of a web service; (ii) specifications related to human-computer interactions, e.g. response times of individual user actions; and (iii) specifications related to business activities, e.g. the number and availability of resources.

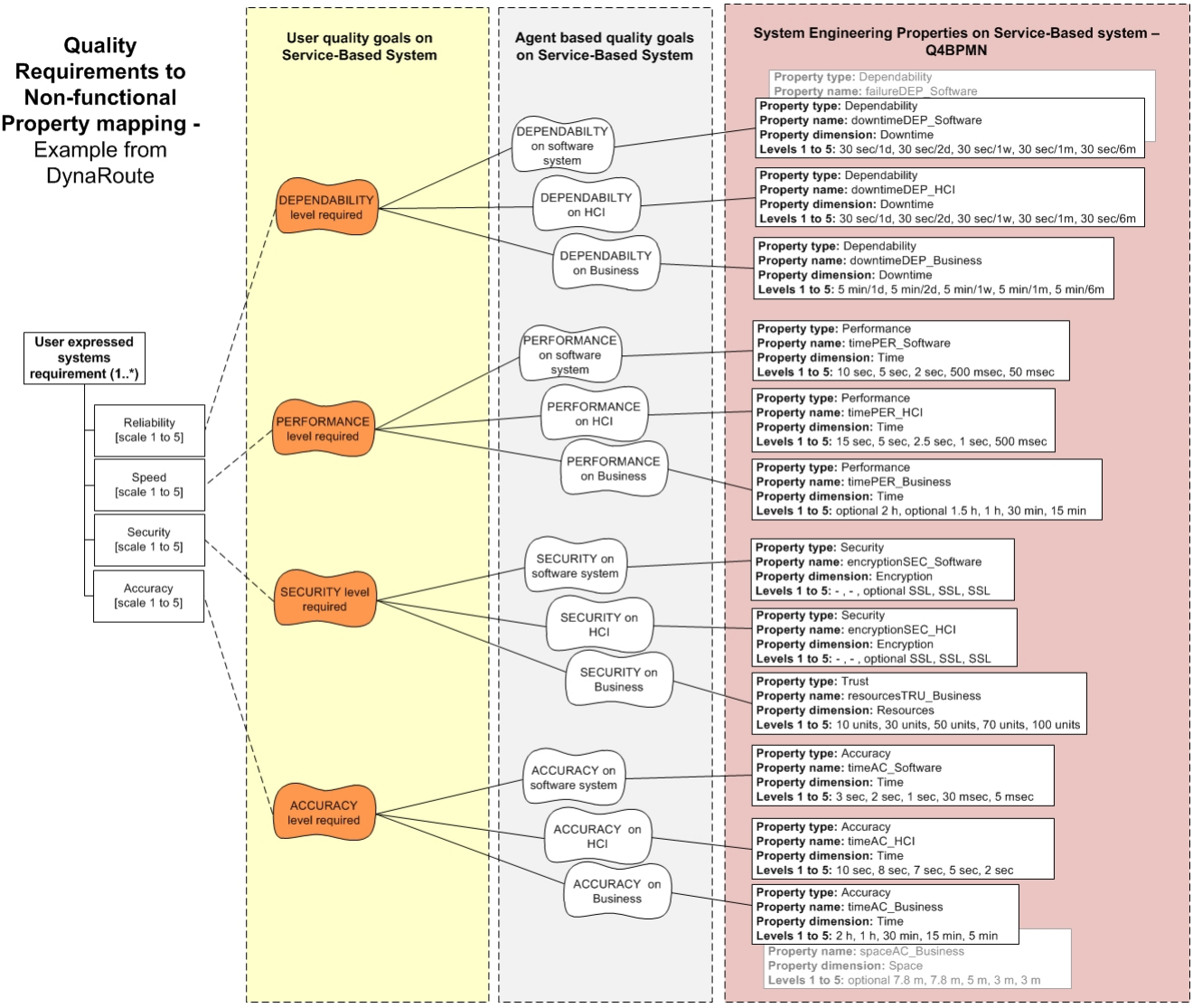


Figure 2.6: Quality requirements mapping to non-functional properties for the DynaRoute example

The qualities and their concerns were mapped to system engineering properties, as defined for Q4BPMN. We have defined four classes of properties in this example: i) dependability properties concerning the availability of the system and failure rates; ii) performance properties related to time, mainly from a software and human interaction point of view; (iii) security properties related to encryption of

operations and trustworthiness of the business activities; and (iv) accuracy properties that relate to time and space dimensions. In order to define the abstract properties, we drew upon research in disciplines such as human-computer interaction and software reliability, for example, we considered the Secure Sockets Layer (SSL) protocol for security encryption-type requirements. We then utilised the domain knowledge available to us from the DynaRoute use case to review the defintions and quantify the predefined ranges.

Returning to the requirements-choreography task matrix, presented in Figure 2.4, one can see that a number of user expressed requirements map onto the choreography task `Request Taxi Service`. These requirements, in particular the non-functional requirements, can be used by the choreography designer as guidance for applying quality properties to the choreography diagram. Figure 2.7 shows an example mapping between 3 original requirements and the quality properties they relate to. Taking the first requirement, the traveller (customer) details are required to be transmitted with a performance level of 5, which translates to the software time performance property. Indeed, the required time performance for all software interactions within this choreography task is therefore quantified as `500msec`. The second and third requirements follow a similar mapping pattern, however the fourth requirement is specified on the participant `Taxi Company`. The requirement specifies that the taxi company shall be reputable, which maps a onto a trust property related to the business security quality goal. Further examples and details of Q4BPMN applied to DynaRoute are provided in deliverable D4.4 [44].

Choreography element, ***Name***	**Original requirement Name, ID, description, quality**	**Quality Property**
Task *Request Taxi Service*	MID send customer details (NFR0012): MID shall be able to transmit traveller (customer) details to the taxi company [Performance 5]	`timePer_Software_L5`
	Reliable Taxi booking (NFR0078): The user shall be able to book a taxi reliably [Reliability 5]	`downtimeDEP_HCI_L5`
	Secure transmission of data (NFR0084): The transmission of user requests shall be secure [Security 4]	`encryptionSEC_Software_L4L5`
Participant property *Taxi Company*	Reputable taxi company (NFR0086): The taxi company shall have an established reputation [Security 4]	`taxiCompanyBusinessTrust`

Figure 2.7: An example of mapping original system requirements to quality properties

2.3. CHOReOS Requirements Tool in Action

We applied the CHOReOS Requirements Tool to the DynaRoute use case and reported the results in deliverable D8.4 [39]. In this section, however, we assess the technical aspects of our tool-based approach and also take a look at the underlying aspects of the tool integration.

2.3.1. Scalability and usability of the tool

For the DynaRoute use case we specified 97 requirements in the Requirements Tool, however, in reality hundreds, if not thousands, could be specified. Our set up of the tool for DynaRoute used the XAMMP open source web server package[3], and in particular the MySQL database. As such, these are scalable databases given the reported file size limits on common operating systems[4]. Another consideration is the number of rows that can be accommodated in the tool's requirements table, implemented in java

[3] `http://www.apachefriends.org/en/xampp.html`
[4] `http://dev.mysql.com/doc/refman/5.0/en/table-size-limit.html`

as a JTable. A JTable with default rowHeight (16) and InterCell spacing (1) is estimated to be able to contain over 120 million rows, as reported by Oracle[5]. This addresses scalability in one sense, however, usability is a factor that affects this. For example, moving the scrollbar by one pixel would move the table by more than a page, therefore preventing many requirements from being displayed. Possible options for improving the scalability of the Requirements Tool include the pagination of results and dynamic loading of entries from the database.

The Requirements Tool provides the domain expert with functions to help them manage and manipulate the large number of entries in the data tables. In particular, two functions are provided to cluster requirements for specifying a single choreography: a keyword search function and the similarity algorithm, mentioned above. Figure 2.8 shows the results from our DynaRoute application after searching for the terms `taxi` and `route`. This is a simple and effective way of sorting the requirements so that those containing the keyword are brought to the top of the table. However, requirements containing the keyword are not necessarily relevant for the cluster. Therefore, the user can run the similarity algorithm on any given requirement. Application of the algorithm generates a measure of semantic similarity between the selected requirement and the other requirements in the data set. Figure 2.9 shows the results of running the algorithm on the requirement `The user shall receive a taxi booking within an acceptable time`. Interesting points to note are that a duplicate requirement was found, as the selected requirement was recorded as a functional and non-functional requirement, and some false double entries were shown in the results table (a correction to the tool is needed here). Unlike the search for `taxi`, the similarity match sorts the table with the most similar and potentially relevant at the top.

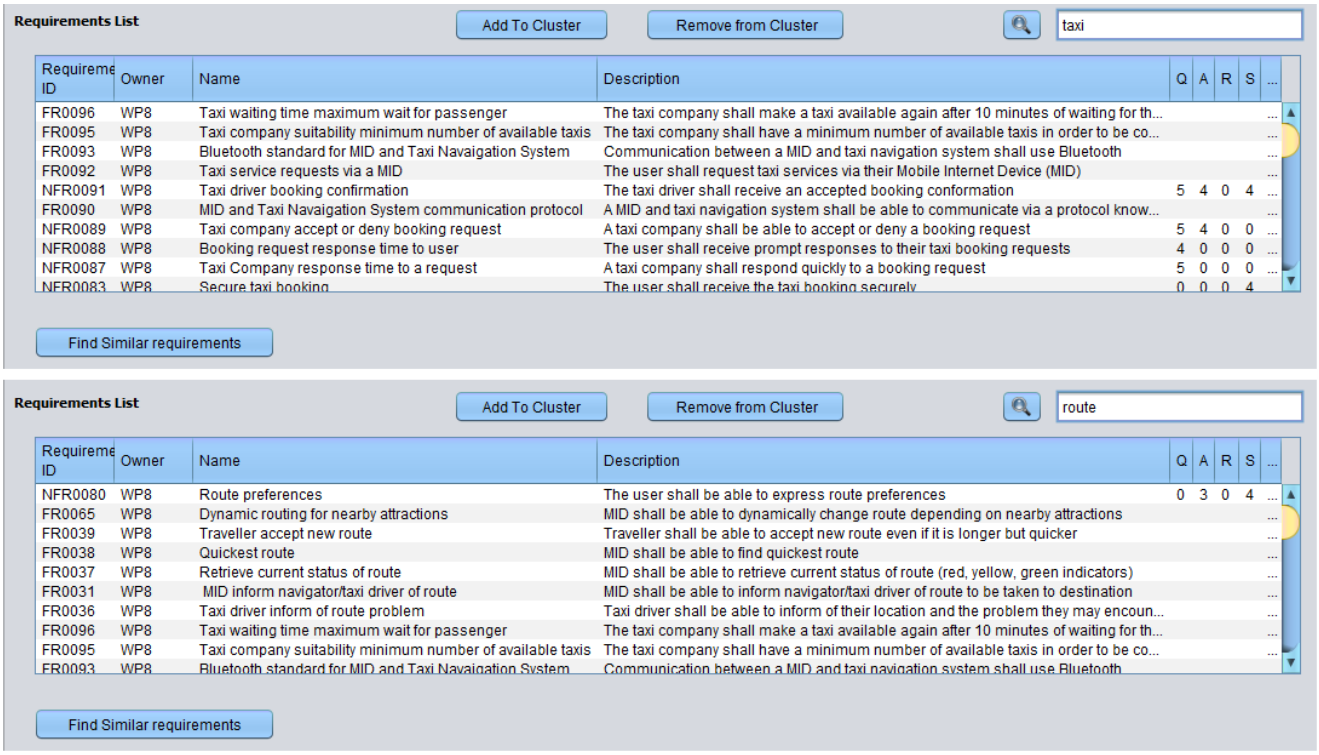
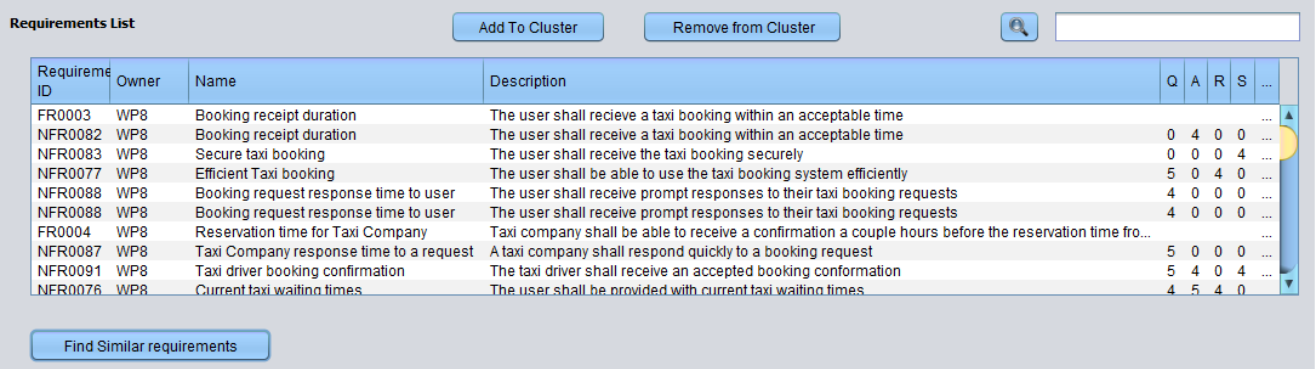


Figure 2.8: CHOReOS Requirements Tool, search results for taxi and route in the requirements list

However, there is a usability concern with this function, as matching the selected requirement to the other 96 requirements took 5 minutes and 19 seconds using a standard laptop on the City University network. A possible option for improving the performance of this function is to store the match results in a requirements matrix. There would be an initial payload, but once the matrix is populated it could be

[5] http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4140598



Requirement ID	Owner	Name	Description	Q	A	R	S	...
FR0003	WP8	Booking receipt duration	The user shall recieve a taxi booking within an acceptable time					...
NFR0082	WP8	Booking receipt duration	The user shall receive a taxi booking within an acceptable time	0	4	0	0	...
NFR0083	WP8	Secure taxi booking	The user shall receive the taxi booking securely	0	0	0	4	...
NFR0077	WP8	Efficient Taxi booking	The user shall be able to use the taxi booking system efficiently	5	0	4	0	...
NFR0088	WP8	Booking request response time to user	The user shall receive prompt responses to their taxi booking requests	4	0	0	0	...
NFR0088	WP8	Booking request response time to user	The user shall receive prompt responses to their taxi booking requests	4	0	0	0	...
FR0004	WP8	Reservation time for Taxi Company	Taxi company shall be able to receive a confirmation a couple hours before the reservation time fro...					...
NFR0087	WP8	Taxi Company response time to a request	A taxi company shall respond quickly to a booking request	5	0	0	0	...
NFR0091	WP8	Taxi driver booking confirmation	The taxi driver shall receive an accepted booking conformation	5	4	0	4	...
NFR0076	WP8	Current taxi waiting times	The user shall be provided with current taxi waiting times	4	5	4	0	

Figure 2.9: CHOReOS Requirements Tool, similarity match results for a selected requirement (listed at the top)

updated for each new or changed requirement. This would remove the need for invoking the similarity algorithm web service for every requirement each time it is run.

A more complete usability assessment of the CHOReOS Requirements Tool is provided in deliverable D10.3 [43]

2.3.2. Integration in action

Once the cluster has been completed, the requirements descriptions are fired at the TEDDiE service to retrieve relevant user task models to help inform the choreography design. The Requirements Tool builds a request text string containing the XML structure, based on an XML schema specified for the third party TEDDiE service, as shown in Figure 2.10. It is sent using the Simple Object Access Protocol, SOAP.

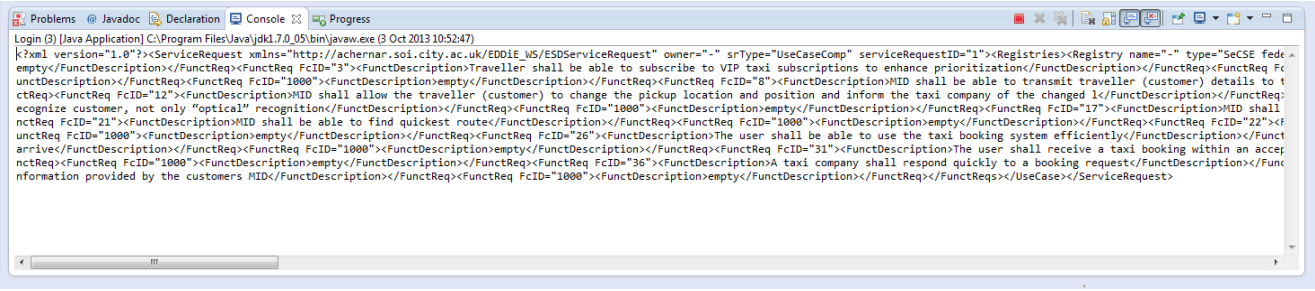


Figure 2.10: TEDDiE request showing the requirements to be fired at the CTT model catalogue

The requirements tool receives the SOAP message response from TEDDiE and extracts the relevant XML text string from the message. Figure 2.11 shows parts of the response received by the tool. Within the response are the retrieved CTT models, explicit trace links to the requirements that matched with each CTT model and associated service classes.

The match value for the `Request taxi` CTT model was 100% which was to be expected given the requirements contained in the requirements cluster. However, the retrieved model `Calculate route` had a surprisingly low match value of 46%, as shown in Figure 2.12. To improve the match performance, a longer and more detailed CTT task description is required.

Once the response is received from TEDDiE, the Requirements Tool parses the XML string and builds a new bespoke XML file for input into MagicDraw. Figure 2.13 shows the file generated in this example from the DynaRoute use case. The file contains:

- Choreography tasks with their underlying CTT user tasks and service classes documented as design guidance.

- BPMN gateways, if applicable from any of the retrieved CTT models.
- Relationships between the choreography elements e.g sequence links between activities.
- Traces between the requirements that matched with each retrieved CTT model.
- The full set of requirements from the cluster, including the original requirements identifiers and quality scores.

Finally, the XML file can be imported into MagicDraw via the Cameo Inter-Op tool. This process is described in detail in deliverable D9.6.2 [40]

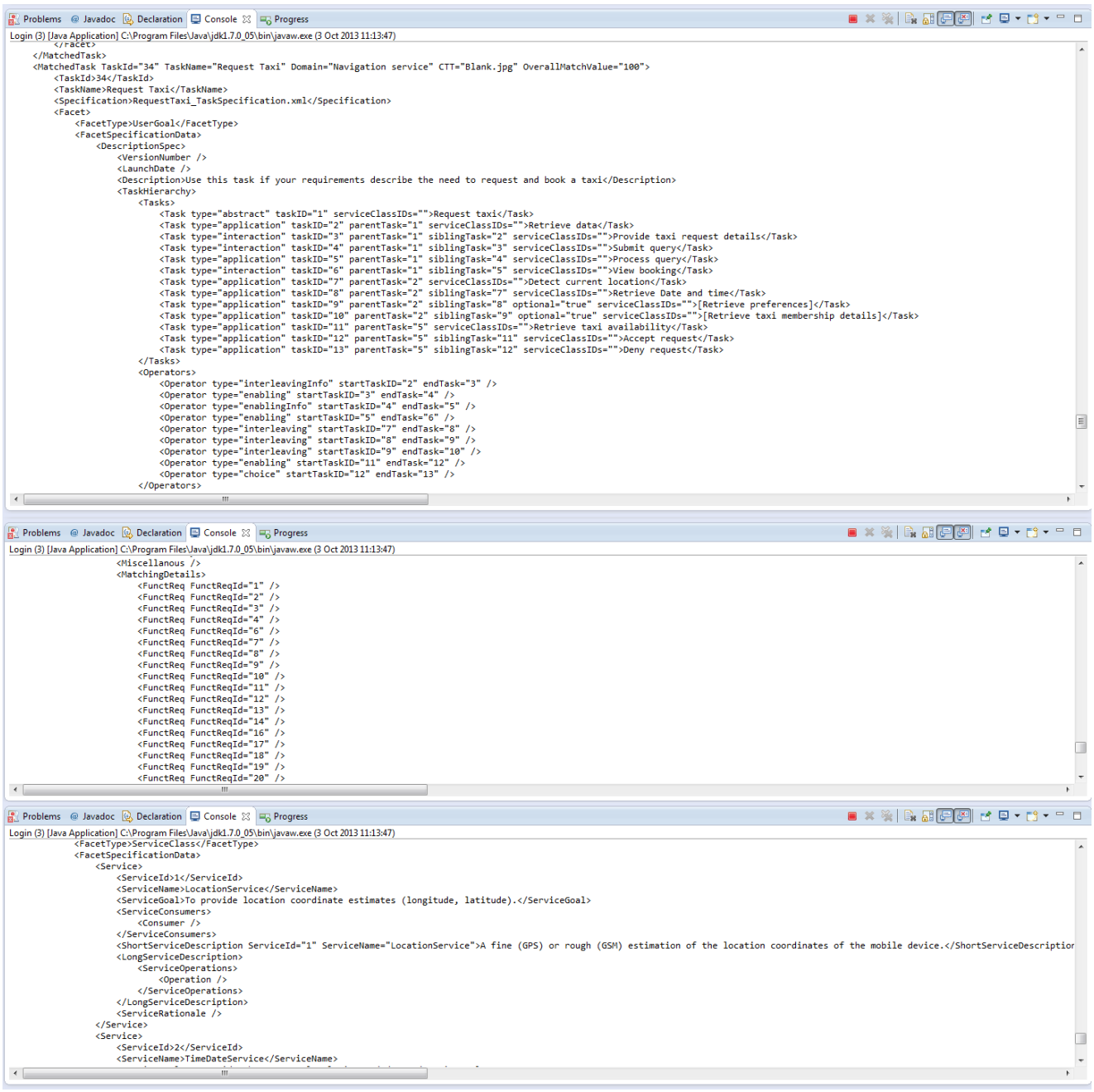


Figure 2.11: TEDDiE response showing a retrieved CTT model, matched requirement IDs and service classes

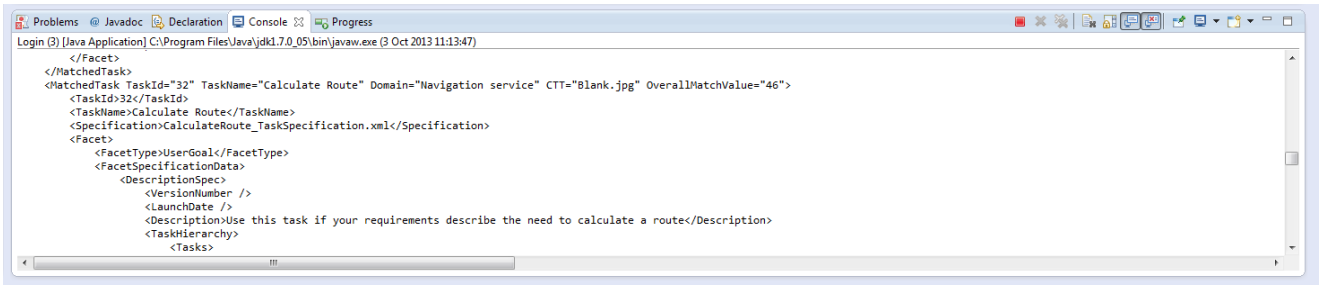


Figure 2.12: TEDDiE response showing the retrieved Calculate route CTT model

```
<?xml version="1.0" encoding="UTF-8"?>
<Choreography name="CoTaxiing">
  - <ChoreographyTasks>
        <ChoreographyTask name="Request route" id="1">CTT tasks: Retrieve input data, Input request details, Submit query, Detect current
            location, Retrieve Date and time, [Retrieve preferences], [Retrieve existing routes], Verify address, Provide preferences. Service
            Classes: LocationService, TimeDateService, RoutingService, AddressService, TrafficDetection, MapData</ChoreographyTask>
        <ChoreographyTask name="Provide route" id="2">CTT tasks: Compute route, View route, Access map services, Retreive traffic data.
            Service Classes: LocationService, TimeDateService, RoutingService, AddressService, TrafficDetection, MapData</ChoreographyTask>
        <ChoreographyTask name="Request taxi booking" id="3">CTT tasks: Retrieve data, Provide taxi request details, Submit query, Detect
            current location, Retrieve Date and time, [Retrieve preferences], [Retrieve taxi membership details]. Service Classes:
            LocationService, TimeDateService, TaxiNotification, UserPreferences</ChoreographyTask>
        <ChoreographyTask name="Deny request" id="4"/>
        <ChoreographyTask name="Accept request" id="5"/>
        <ChoreographyTask name="Provide taxi booking" id="6">CTT tasks: Process query, View booking, Retrieve taxi availability, Accept
            request, Deny request. Service Classes: LocationService, TimeDateService, TaxiNotification, UserPreferences</ChoreographyTask>
        <Gateway name="Accept/deny" id="7" type="Exclusive"/>
    </ChoreographyTasks>
  - <Relationships>
        <Relationship type="sequence" startID="1" endID="2"/>
        <Relationship type="sequence" startID="2" endID="1"/>
        <Relationship type="sequence" startID="3" endID="7"/>
        <Relationship type="sequence" startID="7" endID="4"/>
        <Relationship type="sequence" startID="4" endID="3"/>
        <Relationship type="sequence" startID="7" endID="5"/>
        <Relationship type="sequence" startID="5" endID="6"/>
        <Relationship type="trace" startID="REQ2" endID="1"/>
        <Relationship type="trace" startID="REQ3" endID="1"/>
        <Relationship type="trace" startID="REQ6" endID="1"/>
            Etc ...
        <Relationship type="trace" startID="REQ39" endID="3"/>
    </Relationships>
  - <Requirements>
        <Requirement name="Pre arrange taxi arrival time(FR0003)" id="REQ1">Traveller shall be able to arrange beforehand when the taxi
            shall arrive</Requirement>
        <Requirement name="Reservation time for Taxi Company(FR0004)" id="REQ2">Taxi company shall be able to receive a confirmation a
            couple hours before the reservation time from MID</Requirement>
        <Requirement name="VIP taxi subscriptions(FR0006)" id="REQ3">Traveller shall be able to subscribe to VIP taxi subscriptions to
            enhance prioritization</Requirement>
        <Requirement name="MID location coordinates(FR0008)" id="REQ4">MID shall be able to transmit location coordinates to taxi
            company</Requirement>
        <Requirement name="Taxi type(FR0009)" id="REQ5">Traveller shall be able to choose type of car, big boot, air-conditioning,
            etc.</Requirement>
        <Requirement name="Taxi Driver English language(FR0010)" id="REQ6">Traveller shall be able to request a taxi driver who speaks
            English</Requirement>
        <Requirement name="Taxi green card(FR0011)" id="REQ7">Traveller shall be able to request a taxi with green card</Requirement>
        <Requirement name="MID send customer details(FR0012)" id="REQ8">MID shall be able to transmit traveller (customer) details to the
            taxi company</Requirement>
        <Requirement name="Taxi number(FR0013)" id="REQ9">MID shall be able to display the taxi number that will pick up the traveller
            (customer)</Requirement>
        <Requirement name="Taxi arrival time(FR0014)" id="REQ10">Traveller shall be able to know how long it will take for the taxi to
            arrive</Requirement>
            Etc ...
```

Figure 2.13: XML output file from the CHOReOS Requirements Tool

3 Large scale service base: management, registration and query engine

In this chapter we detail the latest developments concerning the CHOReOS Abstraction-Oriented Service Base Management (AoSBM). The main facilities of AoSBM have been realized as part of WP2; the distribution of the AoSBM software is packaged as part of the overall CHOReOS middleware and specifically the CHOReOS eXtensible Service Discovery (XSD) facility that has been defined in the context of WP3 [29]. This chapter is structured as follows: In Section 3.1, we recall the overall architecture of AoSBM and provide a brief overview of the facilities that have been realized in the first 2 years of CHOReOS [28, 36]. In Section 3.2, we discuss in detail the AoSBM extensions that have been realized in the last year of the project. Finally, in Section 3.3 we provide a qualitative assessment of the usefulness of AoSBM in the CHOReOS use cases. We further note that a quantitative assessment that focuses on the performance of AoSBM in a ULS setting is given in D3.3 [27].

3.1. AoSBM at a Glance

The purpose of AoSBM is to enable service discovery in the context of the FI. In this context, the anticipated growth in the number of available service descriptions should be effectively sustained. To get a more concrete view of the aforementioned growth, in D1.2 [32] we studied several efforts that have been made since 2003 towards quantifying the amount service descriptions that are available through the Internet. According to these efforts, from 2003 till 2009, this amount progressively increased from few hundreds to thousands of service descriptions that could be found by crawling the Web using the Google and the Yahoo search engines, and querying widely known registries. Taking a step towards the FI, the EU FI Assembly vision document [21] estimates that until 2015, the number of available service descriptions will raise up to millions. Beyond 2015, this number is expected to scale up to billions, trillions and so on. Therefore, in the FI a lookup query for services that provide certain functional/nonfunctional properties should be matched against a huge number of available service descriptions. Consequently, the query execution time shall increase with respect to this huge number of service descriptions.

To enable service lookup in the context of FI, the AosBM relies on the concept of *service abstractions*. In particular, we distinguish between *functional* and *non-functional abstractions*. Intuitively, a functional abstraction represents a group of services that offer similar functional properties; it is characterized by an *abstract interface* and a *mapping* between the abstract interface and the interfaces of the represented services. Similarly, a non-functional abstraction represents a group of services that offer similar non-functional properties; it is characterized by an *abstract non-functional description* that consists of a set of layman's terms, one for each non-functional property of interest (e.g., response-time, availability) and a mapping *between* these terms to ranges of concrete values, for each non-functional property of interest. The concept of service abstractions have been formally defined in D1.3 [33] and D1.4 [41]. Moreover, in D2.1 [28] we proposed clustering algorithms that extract abstractions from available service descriptions and exploited the extracted abstractions as a means for organizing similar service descriptions in the AoSBM. To enable efficient service lookup querying, in D2.2 [36] we proposed WSBQL, a query language that is tailored to the concept of abstractions. Roughly a WSBQL specifies functional and non-functional constraints that should be satisfied by the discovered services. The

WSBQL query is matched against abstractions, instead of being matched against service descriptions of concrete services. Therefore, the query execution time scales up with the number of abstractions, instead of scaling up with the number of available service.

Figure 3.1 provides an overview of the main AoSBM facilities.

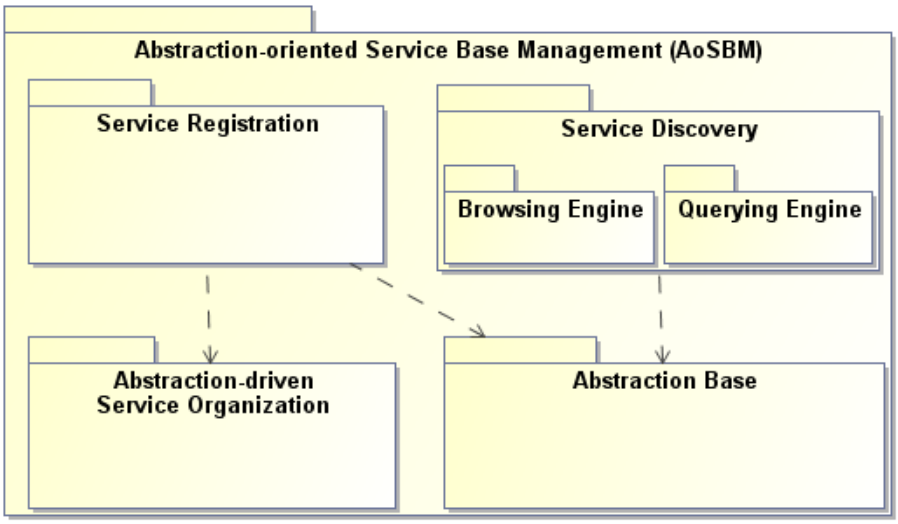


Figure 3.1: The architecture of AoSBM [28].

- The `ServiceRegistration` facility is responsible for populating the AoSBM with information about services, gathered from collections of service descriptions. The collections of service descriptions are provided by the end-user of the AoSBM. Alternatively, the collections can be retrieved via the XSD Plugin Manager [35] that provides unified access to available sources, like the ServicePot that is offered by the Governance and V&V infrastructure. The collections of service descriptions are given as input to the `ServiceRegistration` facility, then they are parsed and transformed to objects that comply with the unified CHOReOS *service model*, which has been formally defined in D1.3 [33] and revised in D1.4 [41].
- The `Abstraction-driven Service Organization` facility realizes the main algorithms that construct hierarchically structured abstractions for a service model that resulted from the registration of a collection of service descriptions to the `ServiceRegistration` facility. The abstractions comply with the CHOReOS *abstractions model*, which has been formally defined in D1.3 [33]. In particular, the `Abstraction-driven Service Organization` facility comprises an hierarchical clustering algorithm that produces clusters of service descriptions, which provide similar functional properties. For each cluster, the hierarchical clustering algorithm constructs a corresponding functional abstraction. Moreover, the `Abstraction-driven Service Organization` facility comprises an partitional clustering algorithm that produces clusters of service descriptions, which provide similar non-functional properties. For each cluster, the partitional clustering algorithm produces a corresponding non-functional abstraction.
- The `AbstractionBase` is the relational store (developed over MySQL) that is used to store information about service descriptions, and service abstractions. The schema of the relational store has been defined in D2.3 [36] and mainly mirrors the service model and the abstractions model, defined in [33, 41].
- The `ServiceDiscovery` facility provides the basic means for exploring the information that is stored in the `AbstractionBase`. In particular, the `BrowsingEngine` facility provides means for browsing service and abstraction related information. The `QueryEngine` accepts as input WSBQL queries, which are matched against the information that is stored in the `AbstractionBase`

and provides as output service and abstraction related information that satisfies the issued queries.

3.2. AoSBM Revised

The main focus of the AoSBM revision was to make the AoSBM facilities available in a distributed setting. To this end, we designed and realized a REST API, named `QueryEngineService`, which exposes the AoSBM `QueryEngine` as a service. Following, we provide further details concerning the design and the functionalities offered by the `QueryEngineService`.

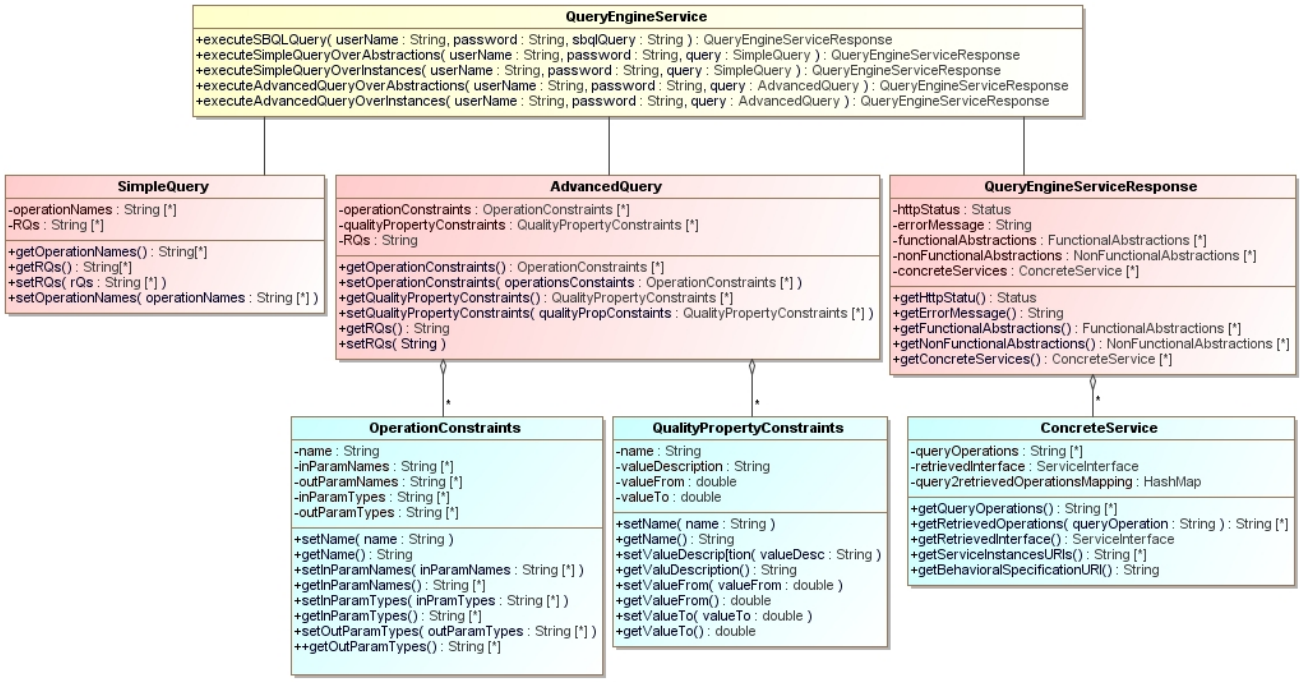


Figure 3.2: Design of the `QueryEngineService`.

Overall, the `QueryEngineService` API provides several operations that can be used for service lookup. In general, these operations accept as input constraints that should be satisfied by the discovered services and produce as output information concerning the discovered services. In general, we divide the operations that are offered by the `QueryEngineService` API in two different categories:

- The operations of the first category allow to use the AoSBM as a typical service registry that does not employ abstractions in the service lookup process Specifically, the first category consists of operations for which the input constraints are matched against the service model information that is stored in the AoSBM.
- The operations of the second category enable abstraction-driven service discovery. In particular, the second category comprises operations for which the input constraints are matched against the abstractions model information that is stored in the AoSBM.

To support typical and more experienced developers, each category provides operations for simple and more advanced lookup queries.

3.2.1. Service Lookup over the Service Model

To enable service lookup over the service model information that is stored in the AoSBM we provide the following alternative options:

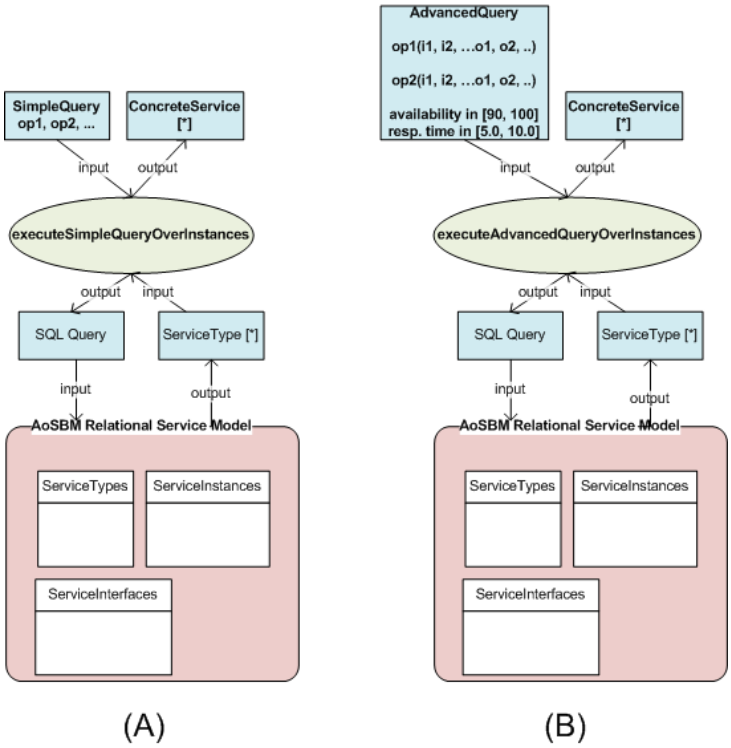


Figure 3.3: Lookup operations over the service model.

1) The `executeSimpleQueryOverInstances`, takes as input a `SimpleQuery` object and produces as output a `QueryEngineServiceResponse` object. The code snippet in Listing 3.1 gives an example of how to call the `executeSimpleQueryOverInstances` and navigate through the results. The `SimpleQuery` object contains the following information:

   - A list of operation names that should match with corresponding names of the operations of the services that will be returned as a result. Specifically, for each required operation name, a discovered service must provide at least one operation, whose name comprises the required operation name.
   - A specification of requirements that concern the services that are used by the discovered services. These requirements may comprise, for instance, the names of the operations that are called by the discovered services.

   The `QueryEngineServiceResponse` object that is produced as output includes a list of `ConcreteService` objects. Each `ConcreteService` object contains information about a discovered service. Specifically, a `ConcreteService` object includes:

   - The list of the required operation names of the input `SimpleQuery` object.
   - The full specification of the `ServiceInterface` that is offered by the discovered service.
   - A mapping between the required operations and the operations of the interface that is offered by the discovered service.

   To facilitate the work of the developer the `ConcreteService` object provides operations that provide easy access to the URIs of the discovered services and to the behavioral specification of the discovered services. These operations reveal the developer from the need to navigate in the `ServiceInterface` object structure.

   The execution of the `executeSimpleQueryOverInstances` operation takes place in three main steps (Figure 3.3(A)):

- Based on the given `SimpleQuery` object, an SQL query is generated over the relations of the AoSBM service model.
- The generated query is executed and a list of `ServiceInterface` objects is reconstructed, based on the information that is retrieved from the AoSBM relational store.
- Finally, the `QueryEngineServiceResponse` that encapsulates the reconstructed `ServiceInterface` objects is constructed and returned to the developer.

```
List<String> operations = new ArrayList<String>();
operations.add("request");
operations.add("get");
String RQs = null;
SimpleQuery simpleQuery = new SimpleQuery(operations, RQs);

Response response = client.executeSimpleQueryOverInstances(simpleQuery);

try {
    InputStream in = (InputStream) response.getEntity();
    JAXBContext context = JAXBContext.newInstance(QueryEngineServiceResponse.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    qeResponse = (QueryEngineServiceResponse) unmarshaller.unmarshal(in);
} catch (WebApplicationException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
} catch (JAXBException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
}

List<ConcreteService> queryResults = qeResponse.getConcreteServices();

for(int i = 0; i < queryResults.size(); i++){
    ConcreteService concreteService = queryResults.get(i);

    result += "\nConcrete Service: " + (i + 1) + "\n\n";
    result += "LTS URI: " + concreteService.getBehavioralSpecificationURI() + "\n";
    result += "ENC URI: " + concreteService.getEnactementURI() + "\n";
    result += "Mappings:" + "\n";

    List<String> queryOperations = null;

    if(simpleQuery != null)
        queryOperations = simpleQuery.getOperationNames();
    else
        queryOperations = advancedQuery.getOperationNames();

    for(int j = 0; j < queryOperations.size(); j++){
        result += "\tQuery operation: " + queryOperations.get(j) + "\n";
        List<Operation> retrievedOperations = concreteService.getRetrievedOperations(queryOperations.
            get(j));

        if(retrievedOperations != null){
            for(int k = 0; k < retrievedOperations.size(); k++){
                result += "\t\t" + retrievedOperations.get(k).getName() + "\n";
            }
        }
    }

    List<String> retrievedURIs = concreteService.getServiceInstancesURIs();
    result += "\nService Instances URIs:" + "\n";
    for(int j = 0; j < retrievedURIs.size(); j++)
        result += "\t" + retrievedURIs.get(j) + "\n";

    result += "\n";
}

System.out.println(underLine + "SimpleQueryOverInstances results" + underLine);
System.out.print(results);
```

Listing 3.1: Executing a simple query over the service model

2) The `executeAdvancedQueryOverInstances`, takes as input an `AdvancedQuery` object and produces as output a `QueryEngineServiceResponse` object. The code snippet in Listing 3.2 gives an example of how to call the `executeAdvancedQueryOverInstances`. The `AdvancedQuery` object contains the following information:

   - A list of `OperationConstraints` objects, which contain functional constraints that should be satisfied by the discovered services. Specifically, an `OperationConstraints` object contains the following information:
     - An operation name that should match with corresponding names of the operations of the services that will be returned as a result. Specifically, a discovered service must provide at least one operation, whose name comprises the required operation name.
     - A list of input (resp. output) parameter names that should match with corresponding input (resp. output) parameter names of the operations of the discovered services. For each required input (resp. output) parameter name, a discovered service must provide at least one operation with an input (resp. output) parameter name that includes the required input (resp. output) parameter name.
     - A list of input (resp. output) parameter types that should match with corresponding input (resp. output) parameter types of the operations of the discovered services. For each required input (resp. output) parameter type, a discovered service must provide at least one operation with an input (resp. output) parameter type that matches with the required input (resp. output) parameter type.
       We assume that the lists have equal number of elements and that elements stored in the same list position correspond to the same required parameter. A list element may be null in case there are no requirements on the name, or the type of the parameter.
   - A list of `QualityPropertyConstraints` objects, which contain non-functional constraints that should be satisfied by the discovered services. Specifically, a `QualityPropertyConstraints` object contains the following information:
     - The name of the quality property of interest.
     - A lower bound for the quality property values of the discovered services.
     - An upper bound for the quality property values of the discovered services.
   - A specification of requirements that concern the services that are used by the discovered services. These requirements may comprise, for instance, the names of the operations that are called by the discovered services.

   As in the case of the `executeSimpleQueryOverInstances`, the `QueryEngineServiceResponse` object that is produced as output from the `executeAdvancedQueryOverInstances` includes a list of `ConcreteService` objects. The execution of the operation takes place as follows (Figure 3.3(B)):

   - Based on the given `AdvancedQuery` object, an SQL query is generated over the relations of the AoSBM service model.
   - The generated query is executed and a list of `ServiceInterface` objects is reconstructed, based on the information that is retrieved from the AoSBM relational store.
   - Finally, the `QueryEngineServiceResponse` that encapsulates the reconstructed `ServiceInterface` objects is constructed and returned to the developer.

```
1 String opName1 = "request";
2
3 List<String> inParamNames1 = new ArrayList<String>();
4 inParamNames1.add("parameters");
5
6 List<String> inParamTypes1 = new ArrayList<String>();
```

```
inParamTypes1.add("string");

List<String> outParamNames1 = new ArrayList<String>();
outParamNames1.add("parameters");

List<String> outParamTypes1 = new ArrayList<String>();
outParamTypes1.add("string");

String opName2 = "get";

List<String> inParamNames2 = new ArrayList<String>();
inParamNames2.add("parameters");

List<String> inParamTypes2 = new ArrayList<String>();
inParamTypes2.add("string");

List<String> outParamNames2 = new ArrayList<String>();
outParamNames2.add("parameters");

List<String> outParamTypes2 = new ArrayList<String>();
outParamTypes2.add("string");

OperationConstraints opConstr1 = new OperationConstraints(opName1, inParamNames1, inParamTypes1,
    outParamNames1, outParamTypes1);
OperationConstraints opConstr2 = new OperationConstraints(opName2, inParamNames2, inParamTypes2,
    outParamNames2, outParamTypes2);

List<OperationConstraints> operationsConstraints = new ArrayList<OperationConstraints>();
operationsConstraints.add(opConstr1);
operationsConstraints.add(opConstr2);

QualityPropertyConstraints qpConstr1 = new QualityPropertyConstraints("ResponseTime", 700, 800);
QualityPropertyConstraints qpConstr2 = new QualityPropertyConstraints("Availability", 40, 50);

List<QualityPropertyConstraints> qualityPropertyConstraints = new ArrayList<
    QualityPropertyConstraints>();
qualityPropertyConstraints.add(qpConstr1);
qualityPropertyConstraints.add(qpConstr2);

AdvancedQuery advancedQuery = new AdvancedQuery(operationsConstraints, qualityPropertyConstraints, null
    );

Response response = client.executeAdvancedQueryOverInstances(advancedQuery);

try {
    InputStream in = (InputStream) response.getEntity();
    JAXBContext context = JAXBContext.newInstance(QueryEngineServiceResponse.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    qeResponse = (QueryEngineServiceResponse) unmarshaller.unmarshal(in);
} catch (WebApplicationException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
} catch (JAXBException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
}

List<ConcreteService> queryResults = qeResponse.getConcreteServices();
```

Listing 3.2: Executing an advanced query over the service model

3.2.2. Service Lookup over the Abstractions Model

To enable service lookup over the abstractions model information that is stored in the AoSBM we provide the following alternatives:

1) The `executeSBQLQuery`, takes as input a WSBQL query and produces as output a `QueryEngineServiceResponse` object. For the detailed syntax and semantics of WSBQL the interested reader may refer to D2.2 [36]. The code snippet in Listing 3.3 gives

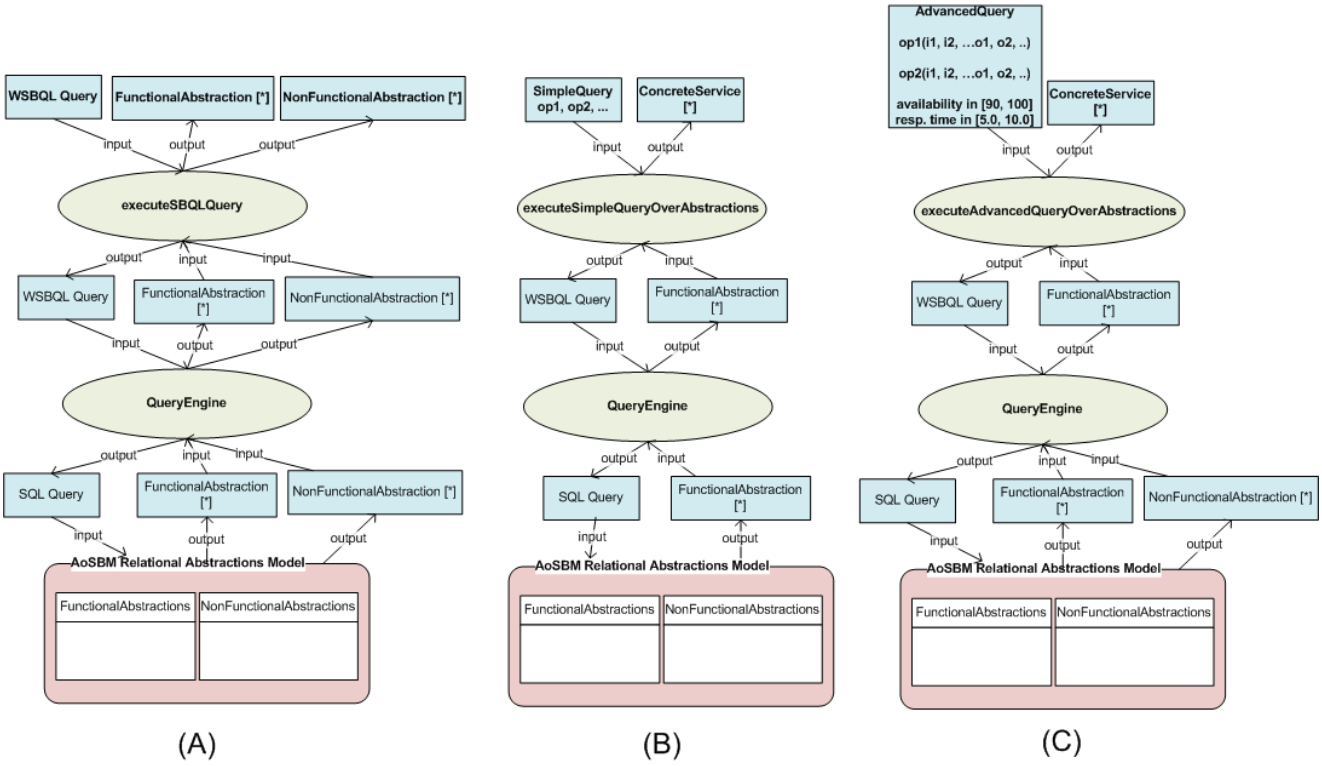


Figure 3.4: Lookup operations over the abstractions model.

an example of how to call the `executeSBQLQuery` and navigate through the results. The `QueryEngineServiceResponse` object that results for the operation may contain the following information:

- A list of `FunctionalAbstraction` objects that contain information regarding the discovered functional abstractions that satisfy the given WSBQL query. The developer may navigate through the information that is included in each `FunctionalAbstraction` object to get the specification of the abstract interface that characterizes the functional abstraction, the specification of the represented service interfaces, the mappings between the abstract interface and the represented service interfaces, etc. For more information concerning the structure of the `FunctionalAbstraction` object the interested reader may also refer to D2.2 [36].
- A list of `NonFunctionalAbstraction` objects that contain information regarding the discovered non-functional abstractions that satisfy the given WSBQL query. Again, the developer may navigate through the information that is included in each `NonFunctionalAbstraction` object to get the specification of the layman's terms that characterize the discovered non-functional abstraction, the represented service instances, etc. Further information concerning the structure of the `NonFunctionalAbstraction` object can be found at D2.2 [36].

The execution of the `executeSBQLQuery` operation takes place in three main steps (Figure 3.4(A)):

- Based on the given `WSBQL` query, an SQL query is generated over the relations of the AoSBM abstraction model.
- The generated query is executed and a list of `FunctionalAbstraction` (resp. `NonFunctionalAbstraction`) objects is reconstructed, based on the information that is retrieved from the AoSBM relational store.

- Finally, the `QueryEngineServiceResponse` that encapsulates the reconstructed `FunctionalAbstraction` (resp. `NonFunctionalAbstraction`) objects is constructed and returned to the developer.

```
String sbqlQuery =

        "let $db = db('localhost/mySB')" + "\n" +
        "for $c in $db/servicecollections" + "\n" +
        "for $fa in $c/hierarchies/abstractions" + "\n" +
        "for $ri in $fa/representativeinterfaces" + "\n" +
        "for $nfa in $c/hierarchies/abstractions" + "\n" +
        "for $pr1 in $nfa/qproperty" + "\n" +
        "where" + "\n" +
                    "$ri/rsi_name like '%orecast%' and" + "\n" +
                    "$pr1/qp_name = 'Availability' and" + "\n" +
                    "$pr1/qp_value = 'High'" + "\n" +
        "return" + "\n" +
                    "Abstractions. fullObject ";

Response response = client.executeSBQLQuery(sbqlQuery);

try {
    InputStream in = (InputStream) response.getEntity();
    JAXBContext context = JAXBContext.newInstance(QueryEngineServiceResponse.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    qeResponse = (QueryEngineServiceResponse) unmarshaller.unmarshal(in);
} catch (WebApplicationException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
} catch (JAXBException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
}

List<FunctionalAbstraction> functionalAbstractions = qeResponse.getFunctionalAbstractions();
List<NonFunctionalAbstraction> nonFunctionalAbstractions = qeResponse.getNonFunctionalAbstractions
    ();

String result = "";

if(functionalAbstractions != null) {
    int faListSize = functionalAbstractions.size();

    if(faListSize != 0) {
        result = "\n---------- Functional Abstractions
            ----------\n\n";
        for(int i = 0; i < faListSize; i++) {
            FunctionalAbstraction fa = functionalAbstractions.get(i);
            result += (i + 1) + "\tinterface name: " + fa.getInterface().getName() + "\n";
        }
    }

}

if(nonFunctionalAbstractions != null) {

    int nfaListSize = nonFunctionalAbstractions.size();

    if(nfaListSize != 0) {
        result += "\n\n---------- Non Functional Abstractions
            ----------\n\n";
        for(int i = 0; i < nfaListSize; i++) {
            NonFunctionalAbstraction nfa = nonFunctionalAbstractions.get(i);
            result += (i + 1) + "\tid: " + nfa.getNonFunctionalAbstractionID() + "\n";
        }
    }

    result += "\n\n";
}
```

Listing 3.3: Executing a WSBQL query over the abstractions model

2) The `executeSimpleQueryOverAbstractions`, takes as input a `SimpleQuery` object and produces as output a `QueryEngineServiceResponse` object. The code snippet in Listing 3.4 gives an example of how to call the `executeSimpleQueryOverAbstractions` and explore the results. The `SimpleQuery` object contains the following information:

   - A list of operation names that should match with corresponding names of abstract operations (i.e., the operations of the abstract interfaces that characterize the discovered functional abstractions), offered by the discovered functional abstractions. For each required operation name, a discovered functional abstraction must provide at least one operation, whose name comprises the required operation name.
   - A specification of requirements that concern the services that are used by the services that are represented by the functional abstractions. These requirements may comprise, for instance, the names of the operations that are called by the represented services.

   The `QueryEngineServiceResponse` object that is produced as output includes a list of `ConcreteService` objects. Each `ConcreteService` object contains information about the services that are represented by the discovered functional abstractions. Specifically, a `ConcreteService` object includes:

   - The list of the required operation names of the input `SimpleQuery` object.
   - The full specification of the `ServiceInterface` that is offered by the represented service.
   - A mapping between the required operations and the operations of the interface that is offered by the represented service.

   The execution of the `executeSimpleQueryOverAbstractions` operation takes place as follows (Figure 3.4(B)):

   - Based on the given `SimpleQuery` object, a WSBQL query is generated over the relations of the AoSBM abstractions model.
   - The generated query is executed and a list of `FunctionalAbstraction` objects is reconstructed, based on the information that is retrieved from the AoSBM relational store.
   - The `ConcreteService` objects that contain information about the services that are represented by the reconstructed `FunctionalAbstraction` objects are created and encapsulated in the `QueryEngineServiceResponse` object.

```
List<String> operations = new ArrayList<String>();
operations.add("request");
operations.add("get");
String RQs = null;
SimpleQuery simpleQuery = new SimpleQuery(operations, RQs);

Response response = client.executeSimpleQueryOverAbstractions(simpleQuery);

try {
    InputStream in = (InputStream) response.getEntity();
    JAXBContext context = JAXBContext.newInstance(QueryEngineServiceResponse.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    qeResponse = (QueryEngineServiceResponse) unmarshaller.unmarshal(in);
} catch (WebApplicationException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
} catch (JAXBException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
}

List<ConcreteService> queryResults = qeResponse.getConcreteServices();

for(int i = 0; i < queryResults.size(); i++){
```

```
25      ConcreteService concreteService = queryResults.get(i);
26
27      result += "\nConcrete Service: " + (i + 1) + "\n\n";
28      result += "LTS URI: " + concreteService.getBehavioralSpecificationURI() + "\n";
29      result += "ENC URI: " + concreteService.getEnactementURI() + "\n";
30      result += "Mappings:" + "\n";
31
32      List<String> queryOperations = null;
33
34      if(simpleQuery != null)
35          queryOperations = simpleQuery.getOperationNames();
36      else
37          queryOperations = advancedQuery.getOperationNames();
38
39      for( int j = 0; j < queryOperations.size(); j++){
40          result += "\tQuery operation: " + queryOperations.get(j) + "\n";
41          List<Operation> retrievedOperations = concreteService.getRetrievedOperations(queryOperations.
                get(j));
42
43          if(retrievedOperations != null){
44              for( int k = 0; k < retrievedOperations.size(); k++){
45                  result += "\t\t" + retrievedOperations.get(k).getName() + "\n";
46              }
47          }
48      }
49
50      List<String> retrievedURIs = concreteService.getServiceInstancesURIs();
51      result += "\nService Instances URIs:" + "\n";
52      for( int j = 0; j < retrievedURIs.size(); j++)
53          result += "\t" + retrievedURIs.get(j) + "\n";
54
55      result += "\n";
56  }
57
58  System.out.println(underLine + "SimpleQueryOverAbstractions results" + underLine);
59  System.out.print(results);
```

Listing 3.4: Executing a simple query over the abstractions model

3) The `executeAdvancedQueryOverAbstractions`, takes as input an `AdvancedQuery` object and produces as output a `QueryEngineServiceResponse` object. The code snippet in Listing 3.5 gives an example of how to call the `executeAdvancedQueryOverAbstractions`. The `AdvancedQuery` object contains the following information:

   - A list of `OperationConstraints` objects, which contain functional constraints that should be satisfied by the abstract interfaces of the discovered functional abstractions. Specifically, an `OperationConstraints` object contains the following information:
     - An operation name that should match with corresponding names of the abstract operations, offered by the discovered functional abstractions. Specifically, a functional abstraction must provide at least one abstract operation, whose name comprises the required operation name.
     - A list of input (resp. output) parameter names that should match with corresponding input (resp. output) parameter names of the abstract operations of the discovered functional abstractions. For each required input (resp. output) parameter name, a discovered functional abstraction must provide at least one operation with an input (resp. output) parameter name that includes the required input (resp. output) parameter name.
     - A list of input (resp. output) parameter types that should match with corresponding input (resp. output) parameter types of the abstract operations of the discovered functional abstractions. For each required input (resp. output) parameter type, a discovered functional abstraction must provide at least one operation with an input (resp. output) parameter type that matches with the required input (resp. output) parameter type.

- A list of `QualityPropertyConstraints` objects, which contain non-functional constraints that should be satisfied by the discovered non-functional abstractions. Specifically, an `QualityPropertyConstraints` object contains the following information:
  - The name of the quality property of interest.
  - A layman's term for the quality property of interest (e.g., high, medium, low)
- A specification of requirements that concern the services that are used by the services that are represented by the discovered functional abstractions. These requirements may comprise, for instance, the names of the operations that are called by the represented services.

The `QueryEngineServiceResponse` object that is produced as output includes a list of `ConcreteService` objects that contain information about the services that are represented by the discovered functional/non-functional abstractions.

The execution of the `executeAdvancedQueryOverAbstractions` operation takes place as follows (Figure 3.4(C)):

- Based on the given `AdvancedQuery` object, a WSBQL query is generated over the relations of the AoSBM abstractions model.
- The generated query is executed and a list of `FunctionalAbstraction` (resp. `NonFunctionalAbstraction`) objects is reconstructed, based on the information that is retrieved from the AoSBM relational store.
- The `ConcreteService` objects that contain information about the services that are represented by the reconstructed objects are created and encapsulated in the `QueryEngineServiceResponse` object. To be present in the `QueryEngineServiceResponse` object, a service must belong in the intersection of (1) the services that are represented by the discovered functional abstraction with (2) the services that are represented by the discovered non-functional abstraction.

```
String opName1 = "request";

List<String> inParamNames1 = new ArrayList<String>();
inParamNames1.add("parameters");

List<String> inParamTypes1 = new ArrayList<String>();
inParamTypes1.add("string");

List<String> outParamNames1 = new ArrayList<String>();
outParamNames1.add("parameters");

List<String> outParamTypes1 = new ArrayList<String>();
outParamTypes1.add("string");

String opName2 = "get";

List<String> inParamNames2 = new ArrayList<String>();
inParamNames2.add("parameters");

List<String> inParamTypes2 = new ArrayList<String>();
inParamTypes2.add("string");

List<String> outParamNames2 = new ArrayList<String>();
outParamNames2.add("parameters");

List<String> outParamTypes2 = new ArrayList<String>();
outParamTypes2.add("string");

OperationConstraints opConstr1 = new OperationConstraints(opName1, inParamNames1, inParamTypes1,
    outParamNames1, outParamTypes1);
OperationConstraints opConstr2 = new OperationConstraints(opName2, inParamNames2, inParamTypes2,
    outParamNames2, outParamTypes2);

List<OperationConstraints> operationsConstraints = new ArrayList<OperationConstraints>();
```

```
33 operationsConstraints.add(opConstr1);
34 operationsConstraints.add(opConstr2);
35
36 QualityPropertyConstraints qpConstr1 = new QualityPropertyConstraints("ResponseTime", 700, 800);
37 QualityPropertyConstraints qpConstr2 = new QualityPropertyConstraints("Availability", 40, 50);
38
39 List<QualityPropertyConstraints> qualityPropertyConstraints = new ArrayList<
       QualityPropertyConstraints>();
40 qualityPropertyConstraints.add(qpConstr1);
41 qualityPropertyConstraints.add(qpConstr2);
42
43 AdvancedQuery advancedQuery = new AdvancedQuery(operationsConstraints, qualityPropertyConstraints, null
       );
44
45 Response response = client.executeAdvancedQueryOverAbstractions(advancedQuery);
46
47 try {
48     InputStream in = (InputStream) response.getEntity();
49     JAXBContext context = JAXBContext.newInstance(QueryEngineServiceResponse.class);
50     Unmarshaller unmarshaller = context.createUnmarshaller();
51     qeResponse = (QueryEngineServiceResponse) unmarshaller.unmarshal(in);
52 } catch (WebApplicationException e) {
53     e.printStackTrace();
54     JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
55 } catch (JAXBException e) {
56     e.printStackTrace();
57     JOptionPane.showMessageDialog(null, e.getMessage(), "error", JOptionPane.ERROR_MESSAGE);
58 }
59
60 List<ConcreteService> queryResults = qeResponse.getConcreteServices();
```

Listing 3.5: Executing an advanced query over the abstractions model

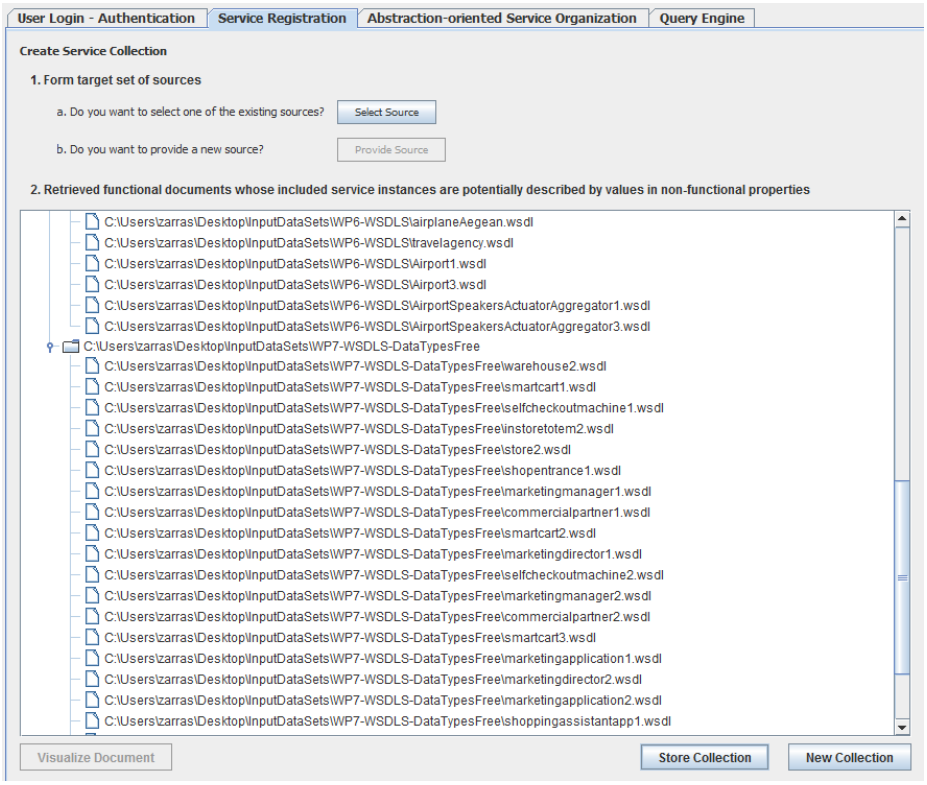


Figure 3.5: AoSBM study - Service registration.

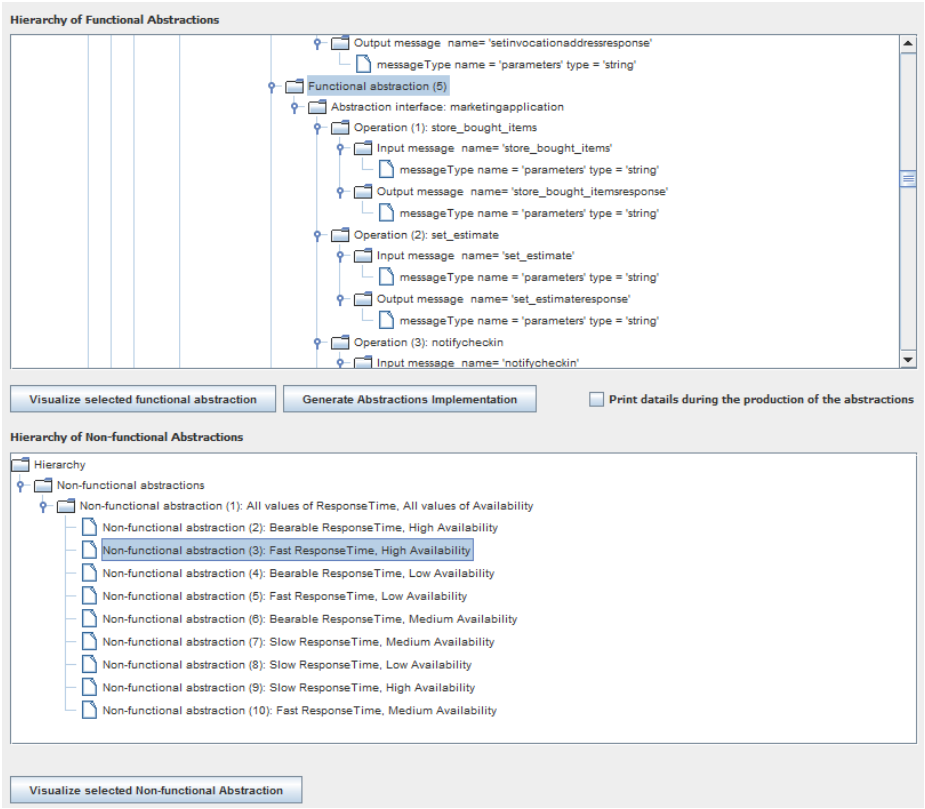


Figure 3.6: AoSBM study - Abstractions-oriented organization.

3.3. AoSBM in Action

The main idea behind AoSBM is to employ service abstractions for the organization of similar services and use the abstraction-oriented organization of services for service lookup. In this section, we assess the usefulness of this idea in the context of the CHOReOS use cases. Our assessment considers the Passenger-Friendly Airport use case and the Adaptive Customer Relationship Booster. Following, we provide the main questions that we investigate towards the AoSBM assessment and discuss the intuition behind these questions.

RQ1 Is the AoSBM able to mine abstractions that are relevant to the different participant roles, specified in the use cases, starting from a collection of service descriptions developed in the context of the use cases? In general, the AoSBM constructs service abstractions based on clustering algorithms, which group similar service descriptions. Hence, the issue behind this question is to assess the effectiveness of the proposed clustering algorithms.

RQ2 Is the AoSBM able to discover services that are relevant to the different participant roles, via queries that are matched against the mined abstractions? The service abstractions that are produced by the AoSBM are characterized by abstract descriptions (abstract interfaces, layman's terms, ranges of non-functional property values [28]), which are mapped to the descriptions of the represented services. Thus, the purpose of this question is to assess whether the abstract descriptions can be used to discover information about concrete services.

Experimental Method: To address RQ1 and RQ2 we gathered a set of service descriptions that specify services, developed for the Passenger-Friendly Airport use case and a set of service descriptions that specify services, developed for the Adaptive Customer Relationship Booster use case. Following, we produced a unified collection of service descriptions, which served as input for our study. The input

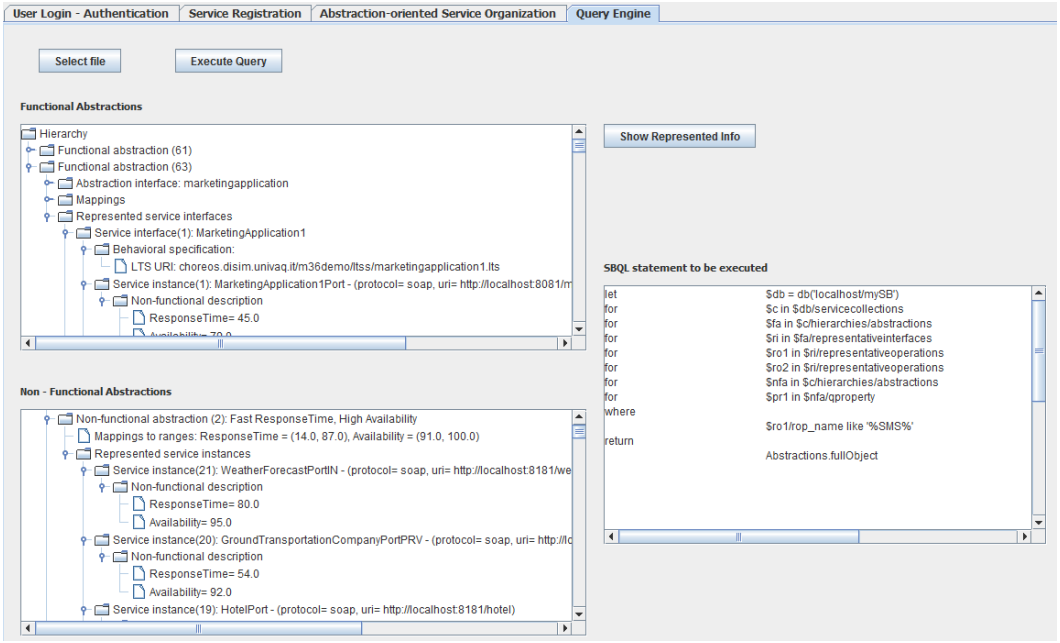


Figure 3.7: AoSBM study - Abstraction-driven service discovery.

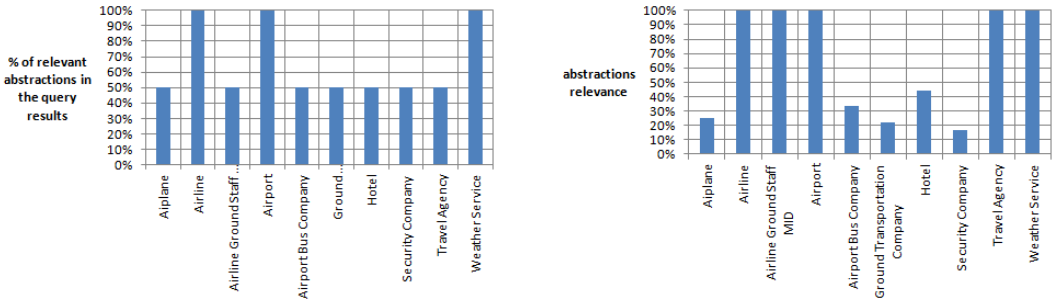
to our study further comprises the choreographies specified in the use cases [31, 30]. Specifically, we focus on the specifications of 20 participant roles, 10 for each use case. The collection of service descriptions that we used in our study contains multiple similar service descriptions that are relevant to each role.

The first step of our study was to register service descriptions to the `ServiceRegistration` facility and produce a corresponding service model, which was stored in the AoSBM relational store (Figure 3.5). The second step of our study was to automatically produce an abstractions model for the registered service descriptions, via the `Abstraction-driven Service Organization` facility (Figure 3.6); the abstractions model was also stored in the the AoSBM relational store. The third step of the study involved the `ServiceDiscovery` facility. In particular, starting from the specifications of the 20 participant roles that we considered, we produced a corresponding set of WSBQL queries. Each query involved a required operation, derived from the specification of the corresponding participant role. The last step of the study was to execute the WSBQL queries (Figure 3.7) and analyze the results with respect to RQ1 and RQ2. The interested reader may find the collections of service descriptions and the queries that have been used in this study in the AoSBM distribution of the CHOReOS project (`(root)/trunk/extensible-service-discovery/AoSBM/src/main/resources/WSDLv1.1/`); the WP6 and the WP7 service descriptions are located in `WP6-WSDLS` and `WP7-WSDLS`, respectively; the WP6 and the WP7 queries are located in `WP6-SBQLQueries` and `WP7-SBQLQueries`, respectively. The study has been performed on a i2 Duo, 2.2 Ghz, 3GB RAM.

Concerning RQ1, we reviewed the abstractions returned from the WSBQL queries that we considered in our study. For each query, the purpose of the review was twofold:

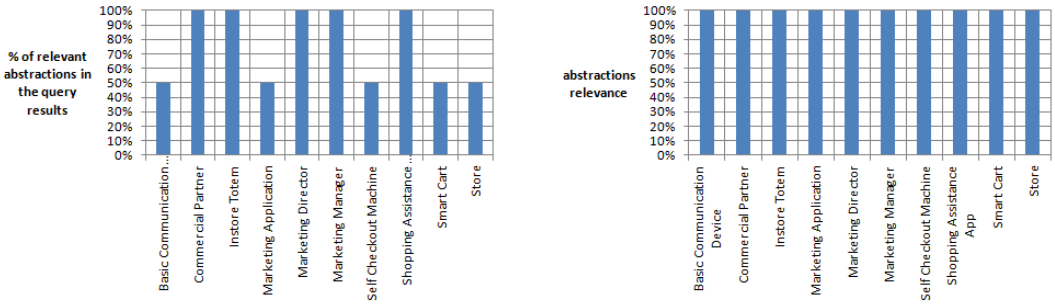
- Our first aim, was *to identify the abstractions that were relevant to the participant role* that we used for deriving the query; we considered that an abstraction is relevant if among its represented services there was at least one that could play the participant role.
- Our second goal, was to quantify the degree to which the abstractions were relevant. To this end, *for each relevant abstraction* we measured *the percentage of represented services that could play the participant role* over the total amount of the represented services. Hereafter, we use the term *abstraction relevance* to refer to this percentage.

Concerning RQ2 we measured the precision and the recall for the WSBQL queries that we consid-



(a) % of relevant abstractions (b) abstractions relevance

Figure 3.8: RQ1 Results - Passenger Friendly Airport.



(a) % of relevant abstractions (b) abstractions relevance

Figure 3.9: RQ1 Results - Adaptive Customer Relationship Booster.

ered in our study. To this end, for each query we identified in the input collection of service descriptions the set of service descriptions, $Relevant$, that are relevant to the query; we considered that a service description is relevant if it can play the participant role that corresponds to the particular query. Moreover, we constructed the overall set of services, $Retrieved$, that are represented by the set of abstractions A, returned from the query. Then, as it is typically done we measured precision and recall as follows:

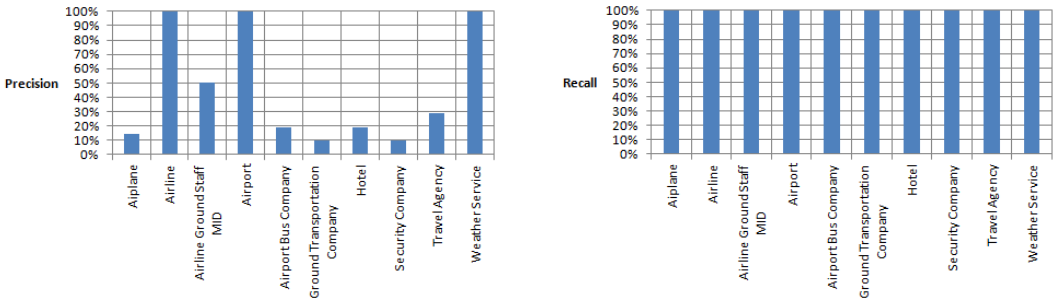
$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|} \tag{3.1}$$

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|} \tag{3.2}$$

Findings: Regarding RQ1, Figures 3.8 and 3.9 summarize the results that we obtained for the Passenger-Friendly Airport and the Adaptive Customer Relationship Booster, respectively. Our main observations are discussed below:

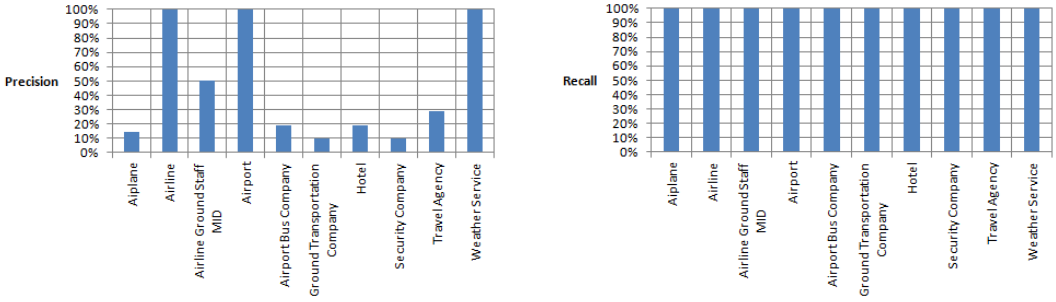
- Overall, in both case studies the percentage of relevant abstractions, discovered for each participant role, varied from 50% to 100%. *Hence, for each participant role the AoSBM did mine relevant abstractions from the initial collection of service descriptions.* In some cases, we discovered multiple abstractions, per role. Some of the abstractions were relevant to the role, while certain other not. The discovery of irrelevant abstractions for a role is due to the fact that the role behavior is similar to some extent with the behavior of other roles. Therefore, the query that we issued for the role matched with the abstractions that represent services that can play the role and with abstractions that represent services that cannot play the role.

- For certain roles of the Passenger-Friendly Airport use case we discovered the same abstraction for multiple roles. The participant roles for which we discovered the same abstraction are very



(a) Precision (b) Recall

Figure 3.10: RQ2 Results - Passenger Friendly Airport.



(a) Precision (b) Recall

Figure 3.11: RQ2 Results - Adaptive Customer Relationship Booster.

similar. Having very similar participant roles means that the services, which play these roles are also very similar. Such services have been been clustered in the same group and represented by the same abstraction. As anticipated, for abstractions that represent services which correspond to different - but very similar - participant roles, the observed relevance was low. Overall, the relevance of the abstractions in this use case varied from 17% to 100%. On the other hand, in the Adaptive Customer Relationship Booster use case, the actual relevance was 100% in all cases. In this use case, we did not have very similar participant roles. Therefore, the services that can play the participant roles were well separated, in different clusters, represented by corresponding abstractions.

Concerning RQ2, the results that we obtained for the Passenger-Friendly Airport and the Adaptive Customer Relationship Booster, are given in Figures 3.10 and 3.11, respectively. Our observations are:

- For both use cases we observe that the recall that we obtained was 100%. *Therefore, queries that are matched against the abstractions are effective towards the discovery of services for the different participant roles that we considered.*
- For the Adaptive Customer Relationship Booster the precision ranged from 50% to 100%. In this case, the low values of precision are due to the discovery of irrelevant abstractions. For the Passenger-Friendly Airport use case the values of precision that we obtained range from 10% to 100%. The low values of precision in this case are because for certain participant roles we discovered irrelevant abstractions and abstractions with low relevance.

4 Choreography Synthesis Processor

This chapter thoroughly refines and extends our previous results reported in Deliverable D1.3 [33] and Deliverable D1.4(b) [41]. In particular, concerning the choreography synthesis process, the initial version of the Choreography LTS (CLTS) presented in Deliverable D1.3 [33] has been significantly extended in Deliverable D1.4(b) [41] with additional constructs to handle more complex constructs of BPMN2 Choreography Diagrams, e.g., conditional exclusive gateways (decision, alternative paths), inclusive gateways (inclusive decision, alternative but also parallel paths), parallel gateways (creation and merging of parallel flows), standard and sequential loops. As better recalled in Section 4.1, a CLTS represents an automata-based specification of the coordination logic "implied" by the choreography and it is automatically produced out of a BPMN2 choreography specification by following a Model-Driven Engineering (MDE) [24] approach. Thus, a CLTS provides an explicit description of the coordination logic that must be distributed onto the synthesized Coordination Delegates to enforce the choreography. The distributed coordination algorithm (initially introduced in Deliverable D1.3 [33]) has also been thoroughly revisited, and in Section 4.2.1 we formalize the new version, which takes (the new version of) a CLTS as input. As a further advance with respect to our previous work mentioned above, in this chapter, we also formally prove correctness of the coordination algorithm with respect to realizing the choreography specified by the CLTS.

4.1. Synthesis Processor at a Glance

When choreography specification is assessed as well formed (according to the CHOReOS development process described in Deliverable D2.2 [36]), and sufficiently detailed to support automation, it can be used for automatically synthesizing further code-based artifacts that are used by the runtime environment. Before describing the approach that CHOReOS has realized to the automatic choreography synthesis, it is worth to briefly discuss what are the problems usually addressed when considering a choreography-based specification C of the system to be realized, where *roles* characterize the "expected" behaviour of the participants: (a) *realizability check*, i.e., whether C can be realized by implementing each participant as specified by the role to be played; and (b) *conformance check*, i.e., whether the global interaction of a set of services, discovered as suitable participants, satisfies C or not. In the research literature many approaches have been proposed to address these problems, e.g., [9, 23, 22, 7, 8].

However when the goal is to actually realize a service choreography by reusing *third-party* services, hence going beyond just checking its effectiveness, a further problem worth to be considered concerns automated choreography *enforcement*. That is, how to coordinate the interactions between those services discovered as suitable participants in order to fit the choreography specification. This requires to distribute and enforce, among the participants, the global coordination logic extracted from the choreography specification. The general problem here is that, although services may have been discovered as suitable participants, their composite interaction may prevent the choreography realization if left uncontrolled (or wrongly coordinated).

CHOReOS proposes a model-driven synthesis approach to generate based on a BPMN 2.0 choreography model an automata-based specification (called CLTS) of the coordination logic "implied" by the choreography. A CLTS provides an explicit description of the allowed sequences of In/Out ser-

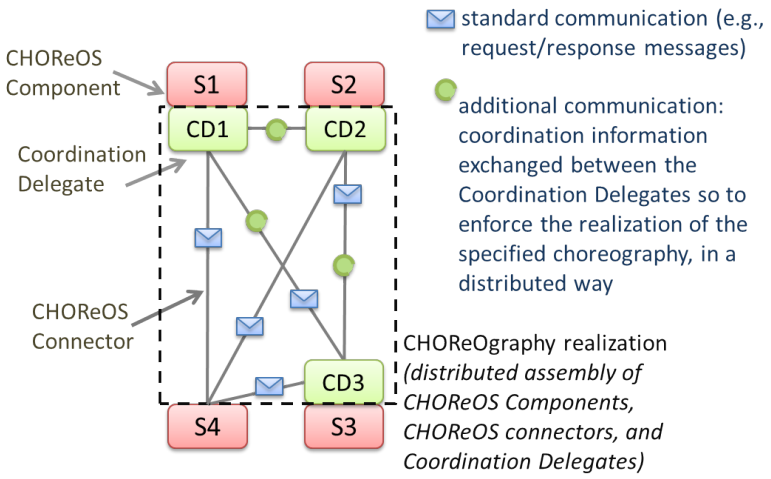


Figure 4.1: CHOReOS architectural style

vice operation calls and, as such, represents an intermediate representation of the choreography that facilitates the extraction of the complex coordination logics "hidden" in BPMN 2.0 choreography specifications [4]. Basically, a CLTS is an extended Labeled Transition System (LTS) that allows handling complex constructs of BPMN 2.0 Choreography Diagrams such as all types of gateways and loops. For the choreography to be externally enforced, the coordination logic modeled by the CLTS is distributed between additional software entities, whose goal is to coordinate (from outside) the interaction of the participant services in a way that the resulting collaboration realizes the specified choreography. The synthesis processor automatically derives these software entities, called Coordination Delegates (CDs), and interposes them among the participant services according to the CHOReOS architectural style (Deliverable D1.4(b) [41]).

The generated CDs are then able to properly access and coordinate the discovered services relying on the communication facilities provided by the CHOReOS middleware.

The coordination logic modelled by C is distributed into a set of *Coordination Models* (CMs), one for each CD. The CDs exploit the CMs in order to coordinate the interaction of the participants in a way that the resulting collaboration realizes the choreography specified by C.

CDs perform pure business-level coordination by intercepting/proxifying the service interaction (i.e., standard communication in Figure 4.1), and mediate it by exchanging the coordination information (i.e., additional communication in the figure) contained in their CMs. In this way, the CDs prevent possible *undesired interactions*. The latter are those interactions that do not belong to the set of interactions allowed by C and can happen when the services collaborate in an uncontrolled way. We formalize the notions of CLTS and CM, hence showing how to decompose a CLTS into a set of CMs. Furthermore, we formalize a distributed coordination algorithm that describes the coordination logic performed by a CD, which relies on its CM. This description is used to generate the actual code implementing a CD.

In order to recall the notion of undesired interactions let us consider the very simple example in Figure 4.2.

The latter shows a BPMN2 choreography specification (a) and its corresponding CLTS (b). We further recall that, in BPMN2, a choreography `Task` is an atomic activity that represents an interaction by means of one or two (request and optionally response) message exchanges between two participants. Graphically, BPMN2 choreography diagrams uses rounded-corner boxes to denote choreography tasks. Each of them is labeled with the roles of the two participants involved in the task (see $p1$ and $p2$), and the name of the service operation (see $op1$ and $op2$) performed by the initiating participant and provided by the other one. A role contained in the white box denotes the initiating participant ($p1$ in the two tasks). In particular, we recall that the BPMN2 specification employs the theoretical concept of a token that, traversing the sequence flows and passing through the elements in a process, aids to

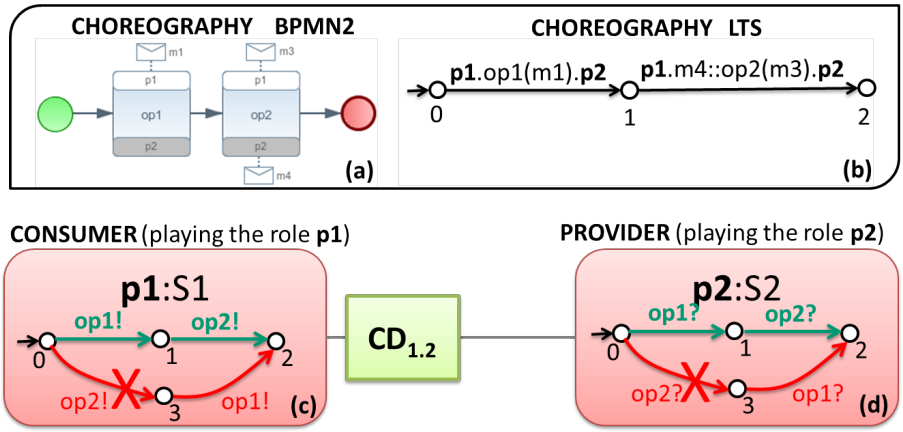


Figure 4.2: Undesired interactions

define its behavior. The start event generates the token that must eventually be consumed at an end event. Basically, the BPMN2 model in the figure specifies that the task $op1$ is followed by the task $op2$. The corresponding CLTS models this sequence of tasks by specifying two contiguous transitions. In particular, the transition label $p1.m4::op2(m3).p2$ specifies that the participant $p1$ initiates the task $op2$ by sending the message $m3$ to the receiving participant $p2$ which, in turn, returns the message $m4$. Let us now assume that $S1$ and $S2$ are the services that have been discovered to play the roles of $p1$ and $p2$, respectively.

Informally, the automaton of $S1$ ($S2$) specifies that $S1$ ($S2$) initiates (receives) $op1!$ ($op1?$) as first, and initiates (receives) $op2!$ ($op2?$) as second, and vice versa. That is, if the services $S1$ and $S2$ interact by following the flow $op1 \rightarrow op2$, the choreography is fulfilled. Vice versa, if the services $S1$ and $S2$ interact by following the flow $op2 \rightarrow op1$, the choreography is not respected. That is, the interaction flow $op2 \rightarrow op1$ is an undesired interaction since, differently from what is specified by the choreography CLTS, the task $op2$ is performed before the task $op1$. As shown in Figure 4.2, a coordination delegate, $CD_{1.2}$, is automatically synthesized and interposed between $S1$ and $S2$ in order to prevent this interaction.

It is worth to mention that the coordination logic performed by the CDs is *service-independent* since it is based on the expected behavior of the participants as specified by C, rather than on the actual one obtained after discovery. Within CHOReOS, this is done to consistently realize *separation of concerns*. That is, to separate pure coordination issues (i.e., undesired interactions) from adaptation/-mediation ones (e.g., syntactic mismatches, data incompatibilities at the service interface level). The latter can arise whenever a service discovered as a participant does not exactly match the role to be played. As described in this chapter, the issue of coordination-based behavioral compatibility is dealt for each choreography that is synthesized, as part of the choreography synthesis process. Moreover, as described in Chapter 3 a form of data mapping based on syntactic matching is dealt with by the large scale service base approach. However, the current implementation of the large scale service base and the synthesis process, does not always ensure mediation-based behavioral and data compatibility. Greater flexibility based on adaptation/mediation can be achieved by considering advanced behavioral compatibility and data compatibility relations. Addressing these issues may include accounting for possible knowledge about the ontology-based semantics of service operations so as to infer n-m mappings between operations and data as investigated within the FP7 ICT Future and Emerging technology project CONNECT (see `https://www.connect-forever.eu/`) [2]. The elicitation of semantic-aware and mediation-based functional abstractions and the definition of mediation/adptation-based choreography synthesis is area for future development of CHOReOS solutions. However, for the sake of completeness, and by continuing the example introduced above, in Section 4.2.4, we show how these issues might be addressed by leveraging our preliminary results reported in [5, 14]. Specifically, we are able to address behavioral mediation and data compatibility issues related to the heterogeneity

of interaction protocols and different granularity/structure of data. For instance, suppose that based on a choreography that we want to synthesize, we discover a service S1 for role R1 and a service S2 for role R2. According to the choreography S1 must send to S2 a message m1. However, S2 requires a message m2 that has a different format from m1. That is, m2 could be obtained either by a portion of m1 or by producing additional data to be merged with m1. As explained in Section 4.2.4, by exploiting ontological information, we are able to automatically synthesize an adaptor that realizes this message conversion and related data mapping. However, if making data compatible means dealing also with issues at the level of the data semantics, e.g., converting the value of a message expressed in meters to the equivalent value expressed in centimetres, specific extra logic might be required and we are not able to deal with that issue automatically. The testing facilities of the CHOReOS Development-Time V&V infratructure might be exploited for the detection of such issues. Then, upon the detection of such issues appropriate adaptors can be developed for certain services. The automated development of such adaptors is a challenging area for future development of the CHOReOS solutions. Another challenging issue is to further combine these results with appropriate data translation facilities like the one that is employs in the CHOReOS middleware to enable service substitution (see Deliverable D3.3 [27]).

4.2. Synthesis Processor Revised

By leveraging the revision of the CLTS model, in this section we formalize the new distributed coordination algorithm initially introduced in Deliverable D1.3 [33], which takes (the new version of) a CLTS as input, and prove its correctness. This allows us to cope with coordination issues. Furthermore, by means of an illustrative example, we also discuss how to deal with possible adaptation issues.

4.2.1. CHOReOS Coordination Protocols: Abstracting Choreography Behavior

Thus, let $\mathcal{T}$ be the universal set of choreography tasks (e.g., service operations), and $\mathcal{M}$ be the universal set of messages. Let $\mathcal{R}$ be the universal set of choreography roles. Let Φ be the universal set of Propositional Logic (PL) formulae. PL formulae are used to distinguish choreography flows depending on whether a specified condition, represented as a *predicate* in PL, holds or not.

Definition 1 (Choreography LTS) *A Choreography LTS (CLTS) C is a tuple $(S_C^{ALT}, S_C^{LOOP}, S_C^{FORK}, S_C^{JOIN}, S_C^{PLAIN}, s_C^0, s_C^f, R_C, T_C, D_C)$, where $S_C^{ALT}, S_C^{LOOP}, S_C^{FORK}, S_C^{JOIN}$ and S_C^{PLAIN} are disjoint and finite sets of states denoting* alternative, loop, fork, join, *and* plain *states, respectively. s_C^0 and s_C^f denote the* initial *(i.e., with no incoming transitions) and* final *(i.e., with no outgoing transitions) states, respectively, and they are neither alternative, loop, fork, join, nor plain states. Let S_C be the union set of all kinds of state, i.e., $S_C = S_C^{ALT} \cup S_C^{LOOP} \cup S_C^{FORK} \cup S_C^{JOIN} \cup S_C^{PLAIN} \cup \{s_C^0, s_C^f\}$. R_C is a finite set of roles, i.e., $R_C \subseteq \mathcal{R}$. $T_C \subseteq (\mathcal{R} \times \mathcal{M} \times \mathcal{T} \times \mathcal{M} \times \mathcal{R}) \cup \Phi$ is a set of transition labels called the alphabet of C. $D_C \subseteq S_C \times T_C \cup \{\varepsilon\} \times S_C$ is the transition relation. ε denotes an ε-transition. C is finite if D_C is finite and C is empty if D_C is empty. We will make use of the following notations: (i) $g \xrightarrow{\alpha}_C h$ iff $(g, \alpha, h) \in D_C \wedge \alpha = r.m_i::t(m_j).r'$ for some $r, r' \in R_C$, $t \in \mathcal{T}$, $m_i, m_j \in \mathcal{M}$; (ii) $g \xrightarrow{\rho}_C h$ iff $(g, \rho, h) \in D_C$ for some $\rho \in \Phi$ or $\rho = |n|$ for some $n \geq 0$; and (iii) $g \longrightarrow_C h$ iff $(g, \varepsilon, h) \in D_C$. We will write that g is a* predecessor *of h or, equivalently, h is a* successor *of g.*

The following properties hold:

- **alternative:** *for all $s^{alt} \in S_C^{ALT}$, s^{alt} has only one predecessor $s \in S_C \setminus \{s_C^f\}$ such that $s \longrightarrow_C s^{alt}$ and s^{alt} has at least two successors $s_1, s_2 \in S_C \setminus \{s_C^0\}$ such that $s^{alt} \xrightarrow{\rho_1}_C s_1$ and $s^{alt} \xrightarrow{\rho_2}_C s_2$ for some $\rho_1, \rho_2 \in \Phi$; for all pairs of successors (s'_i, s'_j), with $i \neq j$, and respective predicates $\rho_i, \rho_j \in \Phi$ such that $s^{alt} \xrightarrow{\rho_i}_C s'_i$ and $s^{alt} \xrightarrow{\rho_j}_C s'_j$ then there is no variable assignment that makes $\rho_i \wedge \rho_j$ hold;*

- **loop:** *for all $s^{loop} \in S_C^{LOOP}$, s^{loop} has only two predecessors $s, s' \in S_C \setminus \{s_C^f\}$ such that $s \longrightarrow_C s^{loop}$ and $s' \longrightarrow_C s^{loop}$, and s^{loop} has only two successors $q, q' \in S_C \setminus \{s_C^0\}$ such that $s^{loop} \longrightarrow_C q$ and $s^{loop} \xrightarrow{\rho}_C q'$*

for some $\rho \in \Phi$; we call q' and q the loop entry state *and* exit state*, respectively;*

- **fork:** *for all $s^{fork} \in S_C^{FORK}$, s^{fork} has only one predecessor $s \in S_C \setminus \{s_C^f\}$ such that $s \longrightarrow_C s^{fork}$ and s^{fork} has at least two successors $s', s'' \in S_C \setminus \{s_C^0\}$ such that $s^{fork} \longrightarrow_C s'$ and $s^{fork} \longrightarrow_C s''$;*
- **join:** *for all $s^{join} \in S_C^{JOIN}$, s^{join} has only one successor $s \in S_C \setminus \{s_C^0\}$ such that $s^{join} \longrightarrow_C s$ and s^{join} has at least two predecessors $s', s'' \in S_C \setminus \{s_C^f\}$ such that $s' \longrightarrow_C s^{join}$ and $s'' \longrightarrow_C s^{join}$.*

Due to the introduction of ε-transitions, in general, a CLTS is non-deterministic. ε-transitions are introduced for dealing with non-plain states both structurally and semantically. For instance, they can be used to specify cascading fork states or to model the (internal) event of creating parallel flows, which originate from a fork state.

From hereon let $C = (S_C^{ALT}, S_C^{LOOP}, S_C^{FORK}, S_C^{JOIN}, S_C^{PLAIN}, s_C^0, s_C^f, R_C, T_C, D_C)$ be a CLTS.

Definition 2 (Flow) *The sequence $t = \alpha_{i+1}\alpha_{i+2}\ldots\alpha_n$ is a* flow *of C iff there exists a sequence of states $s_i \ldots s_n \in S_C$ such that $s_i \xrightarrow{\alpha_{i+1}}_C s_{i+1} \ldots s_{n-1} \xrightarrow{\alpha_n} s_n$, $i \geq 0$, $n > i$. The* empty flow *is denoted by $\ddot{\varepsilon}$. We will make use of the following notation: $s_1 \overset{\alpha}{\Longrightarrow}_C s_n$ iff there exists $n \geq 2$ such that $s_1 \xrightarrow{\alpha}_C s_2 \longrightarrow_C \ldots \longrightarrow_C s_n$.*

Let $1, \ldots, n$ be identifiers for n participant services playing the roles $r_1, \ldots, r_n \in \mathcal{R}$, respectively. Then, the *coordination model* $Coord_{i,j}$, for the operations required by the participant i and provided by the participant j ($i \neq j$), is generated out of C as follows:

$$Coord_{i,j} =$$
$$\{\langle s, t, s', CD_{s'}, true, Notify_s, Wait_{siblings(s)}\rangle \mid s \xrightarrow{r_i.t.r_j}_C s'\} \bigcup$$
$$\{\langle s, \varepsilon, s', CD_{s'}, true, Notify_s, Wait_{siblings(s)}\rangle \mid \exists s^{prec}, t\colon s^{prec} \overset{r_i.t.r_j}{\Longrightarrow}_C s \longrightarrow_C s'\} \bigcup$$
$$\{\langle s, \varepsilon, s', CD_{s'}, \rho_{s.s'}, Notify_s, Wait_{siblings(s)}\rangle \mid \exists s^{prec}, t\colon s^{prec} \overset{r_i.t.r_j}{\Longrightarrow}_C s \xrightarrow{\rho_{s.s'}}_C s'\}$$

where

$$CD_{s'} = \{(h,k) \mid (h,k) \neq (i,j) \wedge \exists s'', t\colon s' \xrightarrow{r_h.t.r_k}_C s''\} \bigcup$$
$$\{(h,k) \mid (h,k) \neq (i,j) \wedge \exists \rho \in \Phi, s'', s^{succ}, t\colon s' \xrightarrow{\rho}_C s'' \xrightarrow{r_h.t.r_k}_C s^{succ}\} \bigcup$$
$$\{(h,k) \mid (h,k) \neq (i,j) \wedge s' \in S_C^{FORK} \cup S_C^{JOIN} \wedge \exists s'', s^{succ}, t\colon s' \longrightarrow_C s'' \xrightarrow{r_h.t.r_k} s^{succ}\}$$

$$Notify_s = \{[s, (h,k)] \mid s' \in S_C^{JOIN} \wedge \exists s'' \neq s, s^{prec}, t : s^{prec} \overset{r_h.t.r_k}{\Longrightarrow}_C s'' \longrightarrow_C s'\}$$

$$Wait_{siblings(s)} = \{[s'', (h,k)] \mid s' \in S_C^{JOIN} \wedge \exists s'' \neq s, s^{prec}, t : s^{prec} \overset{r_h.t.r_k}{\Longrightarrow}_C s'' \longrightarrow_C s'\}$$

Each tuple in $Coord_{i,j}$ is composed of seven elements:

- The **first element** denotes the CLTS source state from which the related CD can either perform the operation specified as **second element** of the tuple or take a move without performing any operation (i.e., the CD can step over an ε-transition). In both cases, the **third element** denotes the reached target state.
- The **fourth element** contains the set of (identifiers of) those CDs whose supervised services became active in the target state, i.e., the ones that will be allowed to require some operation from the target state. This information is used by the "currently active" CD(s) to inform the set of "to be activated" CDs (in the target state) about the changing global state.
- The **fifth element** reports the PL predicate whose validity has to be checked to select the correct tuple, and hence the correct flow(s) in the CLTS.

- The **sixth element** contains the predecessor of a join state that a CD, when reaching it, must notify to the other CDs in the parallel flow(s) of the same originating fork. Complementary, the **seventh element** contains the predecessors of join states that must be waited for.

4.2.2. Distributed coordination algorithm

In this section we formalize a distributed coordination algorithmic for the enforcement of service choreographies, which describes the coordination logic that a CD has to perform by relying on its CM.

The algorithm is defined by the following rules that each delegate $CD_{i,j}$ follows in a distributed setting, when its supervised service S_i sends a request to perform a task t with S_j, without relying on any central synchronizing entity or shared memory. These rules locally characterize the collaborative behavior of the CDs at run-time from a clear *one-to-many* point of view. The most important aspect here is that, upon reaching a join state, CDs wait for and notify each other in order to synchronize their execution. To this end, each $CD_{i,j}$ maintains its own *notify queue* $NQ_{i,j}$ (i.e., the queue of NOTIFY messages) that is unknown to the others CDs. At run time, NOTIFY messages together with a simple notion of *priority* $P_{i,j}$ (initially associated to each $CD_{i,j}$) are used to realize join states as distributed synchronization points. Then, when a join state has been realized, and hence all the parallel executions have been synchronized, the CD with highest priority is in charge of notifying the CDs that, according to the choreography, are allowed to proceed. Each $CD_{i,j}$ sends NOTIFY messages according to the coordination information contained by the sixth element of the $Coord_{i,j}$ tuples. The "co-related" WAIT primitive is instead performed according to the information contained in the seventh element of the tuples.

CDs are *reactive* entities; that is, at each iteration of the algorithm, a CD waits for either its supervised service to make a request or another CD to forward a request.

Beyond using NOTIFY messages, a $CD_{i,j}$ uses the so called UPDATE_CURRENT_STATE messages whenever the current global state of C changes according to the performed coordination. By considering the coordination information represented by the fourth element of its tuples, $CD_{i,j}$ sends an UPDATE_CURRENT_STATE message in order to inform, about the state change, all the other CDs whose execution can progress from the new current global state. Thus, if a $CD_{h,k}$ receives a request to perform a task t while its local execution is in a state where t is not allowed, then it waits for receiving an UPDATE_CURRENT_STATE message on a state from which t is allowed.

The actions defined by each rule are assumed to form a single event (i.e., each rule has to be considered as atomic). Within the rules, we denote with s the current state of the CLTS C. Moreover, we denote with $Coord_{i,j}[\texttt{sourceState}]$ (resp., $Coord_{i,j}[\texttt{targetState}]$) the state of C that is a source (resp., target) state for the transition labeled with $Coord_{i,j}[\texttt{allowedOperation}]$; $Coord_{i,j}[\texttt{allowedOperation}]$ is the operation that can be performed by S_i when C is in the state $Coord_{i,j}[\texttt{sourceState}]$; $Coord_{i,j}[\texttt{allowedServiceInTargetState}]$ is the set of CDs (and, hence, of supervised services) different from $CD_{i,j}$ (and, hence, from S_i) that, with respect to C, are allowed to proceed from $Coord_{i,j}[\texttt{targetState}]$; $Coord_{i,j}[\texttt{predicate}]$ is the PL predicate associated to an alternative path, which allows a CD to proceed on that path whenever it holds; $Coord_{i,j}[\texttt{siblingServiceToBeNotified}]$ (resp., $Coord_{i,j}[\texttt{siblingServiceToBeWaitedFor}]$) is the set of CDs different from $CD_{i,j}$ that, progressing on parallel paths with respect to $CD_{i,j}$, have to be notified (resp., waited for) whenever $CD_{i,j}$ is going to reach the join state for those paths.

Rule 1: Upon receiving, from S_i, a request to perform a task t with S_j in the current state s of C,

1.1 if does not exist $tuple \in Coord_{i,j}$ *s.t.* $tuple[\texttt{allowedOperation}] = t$ (i.e., t is not in the alphabet of C) **then** $CD_{i,j}$ raises an exception UNKNOWN_TASK [1];

[1] Concerning unknown tasks, in the previous version of the algorithm [4] we allowed supervised services to perform them hence adopting a permissive approach only. Now, the UNKNOWN_TASK exception can be handled to be either permissive or restrictive depending on the application context and needs.

1.2 **if** there exists $tuple \in Coord_{i,j}$ ***s.t.*** $tuple[\texttt{sourceState}] = s$ and $tuple[\texttt{allowedOperation}] = t$ (i.e., t is allowed from s) **then**

1.2.1 $CD_{i,j}$ forwards to S_j the message initiating t (hence synchronizing with S_i) [2];

1.2.2 $CD_{i,j}$ updates s to $tuple[\texttt{targetState}]$;

1.2.3 **for every** $(h,k) \in tuple[\texttt{allowedServiceInTargetState}]$, $CD_{i,j}$ sends UPDATE_CURRENT_STATE($tuple[\texttt{targetState}]$) to $CD_{h,k}$;

else if $tuple[\texttt{sourceState}] \neq s$ for each $tuple \in Coord_{i,j}$ ***s.t.*** $tuple[\texttt{allowedOperation}] = t$ (i.e., t is not allowed from s), **then** $CD_{i,j}$ WAITs for receiving UPDATE_CURRENT_STATE($tuple[\texttt{sourceState}]$) [3];

1.3 **if** there exists $tuple \in Coord_{i,j}$ ***s.t.*** $tuple[\texttt{sourceState}] = s$ and $tuple[\texttt{allowedOperation}] = \tau.\varepsilon.\tau$ (i.e., it is possible to proceed from s only by an internal task) **then**

1.3.1 **if** $tuple[\texttt{targetState}] \in S_C^{JOIN}$ **then**

1.3.1.1 **for every** $[s,(h,k)] \in tuple[\texttt{siblingServiceToBeNotified}]$, $CD_{i,j}$ sends NOTIFY(s, $CD_{h,k}$, $tuple[\texttt{targetState}]$) to $CD_{h,k}$;

1.3.1.2 **for every** $[s'',(h,k)] \in tuple[\texttt{siblingServiceToBeWaitedFor}]$ ***s.t.*** $[s'',(h,k)]$ is "*valid*" (i.e., s'' is not reachable from an alternative state whose condition does not hold or from a join state involving $CD_{h,k}$ with a lower priority), $CD_{i,j}$ WAITs for receiving NOTIFY(s'', $CD_{h,k}$, $tuple[\texttt{targetState}]$) from $CD_{h,k}$;

1.3.2 **if** $tuple[\texttt{targetState}] \in S_C^{ALT}$ and there exists $tuple' \in Coord_{i,j}$ ***s.t.*** $tuple'[\texttt{sourceState}] = tuple[\texttt{targetState}]$ and $tuple'[\texttt{predicate}]$ holds (i.e., a valid alternative path exists) **then**

1.3.2.1 $CD_{i,j}$ updates s to $tuple'[\texttt{targetState}]$;

1.3.2.2 **for every** $(h,k) \in tuple'[\texttt{allowedServiceInTargetState}]$, $CD_{i,j}$ sends UPDATE_CURRENT_STATE($tuple'[\texttt{targetState}]$) to $CD_{h,k}$;

else if $tuple[\texttt{targetState}] \in S_C^{ALT}$ and does not exists such $tuple'$ above (i.e., for all $tuple'$, $tuple'[predicate]$ does not hold) **then** $CD_{i,j}$ raises an exception NO_VALID_ALTERNATIVE_PATH [4];

1.3.3 **if** $tuple[\texttt{targetState}] \in S_C^{LOOP}$ **then**

1.3.3.1 **if** $tuple[\texttt{targetState}]$ is visited for the first time [5] **then** let $tuple' \in Coord_{i,j}$ be ***s.t.*** $tuple'[\texttt{sourceState}] = tuple[\texttt{targetState}]$ and $tuple'[\texttt{targetState}]$ is the loop entry state;
else if $tuple[\texttt{targetState}]$ is not visited for the first time [6] **then** let $tuple' \in Coord_{i,j}$ be ***s.t.*** $tuple'[\texttt{sourceState}] = tuple[\texttt{targetState}]$ and $tuple'[\texttt{predicate}]$ holds;

1.3.3.2 $CD_{i,j}$ updates s to $tuple'[\texttt{targetState}]$;

1.3.3.3 **for every** $(h,k) \in tuple'[\texttt{allowedServiceInTargetState}]$, $CD_{i,j}$ sends UPDATE_CURRENT_STATE($tuple'[\texttt{targetState}]$) to $CD_{h,k}$;

Rule 2: When $CD_{i,j}$ receives UPDATE_CURRENT_STATE($state$) for some $state$ that it was waiting for, **then**

[2] In this respect, it is worth to note that (*i*) the service-to-CD interaction is synchronous (i.e., the act of intercepting/delegating is synchronous without any restriction on weather the business-level communication is synchronous or asynchronous); (*ii*) the CD-to-CD interaction is synchronous or asynchronous (without any restriction) depending on the type of the business-level communication among the services; (*iii*) the CD-to-CD exchange of coordination information is asynchronous. Note that, according to the notions of choreography realizability decidability in [8] (and references therein), having a synchronous interception/delegation of service-to-CD interaction does not represent a limitation.

[3] In other words, $CD_{i,j}$ is in a source state, say s', different from s and it is waiting for being notified for s' becoming the new current state of C.

[4] The specified choreography cannot be enforced.

[5] According to the notion of tasks marked as loops in the BPMN2 specification, the body of the loop in our CLTS is executed at least once. The guard is evaluated afterwards.

[6] Either the loop guard is true and, hence, the body of the loop is executed or it is false and, hence, the execution proceeds from the loop exit state.

2.1 $CD_{i,j}$ updates s to $state$;

2.2 **if** a task t is pending [7] **then goto** Rule 1.2;

Rule 3: When $CD_{i,j}$ receives NOTIFY($predecessor\_state$, $CD_{h,k}$, $join\_state$), for some $predecessor\_state$ predecessor of $join\_state$ and some $CD_{h,k}$, **then**

3.1 $CD_{i,j}$ puts NOTIFY($predecessor\_state$, $CD_{h,k}$, $join\_state$) in $NQ_{i,j}$;

3.2 **if** $NQ_{i,j}$ contains all NOTIFY messages that $CD_{i,j}$ was waiting for, and $P_{i,j}{>}P_{h,k}$ for all $CD_{h,k}$ $(h,k{\neq}i,j)$ in the NOTIFY messages in $NQ_{i,j}$ **then**

3.2.1 $CD_{i,j}$ updates s to $join\_state$;

3.2.2 let $tuple \in Coord_{i,j}$ be *s.t.* $tuple[\texttt{sourceState}] = join\_state$;

3.2.3 **for every** $(h,k){\in}tuple[\texttt{allowedServiceInTargetState}]$, $CD_{i,j}$ sends UPDATE_CURRENT_STATE($tuple[\texttt{targetState}]$) to $CD_{h,k}$;

4.2.3. Correctness

In this section, we prove that our distributed coordination algorithm prevents undesired interactions. The contribution of formalizing such a proof, beyond stating correctness of our choreography synthesis method, is that we give a rigorous characterization of the notion of undesired interaction, which in our previous work has been informally treated hence making it difficult to effectively assess our method.

At the beginning of this chapter, we informally write that the undesired interactions are those interactions that do not belong to the set of the interactions modelled by the CLTS. This means that an undesired interaction can occur whenever either (i) a CD receives a request to perform a task that is not allowed in the current state or the latter (from which the execution of the required task is allowed) is such that: (ii) it has been reached by an alternative state whose condition does not hold; (iii) it is a loop entry state and the guard of the loop does not hold; (iv) it is a loop exit state and the guard of the loop holds; (v) it is the successor of a join state although there is still some parallel flow to be completed; or (vi) after performing the required task, it leads to a join state where each CD, involved in a parallel flow, is waiting for the others CDs to complete their parallel flows. Note that conditions (ii), (iii), and (iv) are due to possible run-time faults in evaluating the PL predicates associated to alternative and loop states; whereas conditions (iv) and (v) relate to run-time issues that can arise whenever NOTIFY messages are not suitably waited for and sent, respectively. These considerations lead to the following definition that characterizes the way undesired interactions can be detected at run-time. That is, an undesired interaction occurs whenever an *undesired operation*, as defined by Definition 3, occurs. In the definition, we make use of the following notations. $Coord_{i,j}(s)$ denotes the set of tuples in $Coord_{i,j}$ with s as first element. $Coord_{i,j}(s){\restriction_{Op}}$ denotes the set of choreography operations (different from ε) appearing as second element of tuples in $Coord_{i,j}(s)$. $Coord_{i,j}(s){\restriction_{Wait}}$ denotes the union set of the sets of valid triples appearing as seventh element of tuples in $Coord_{i,j}(s)$. $\sim NQ_{i,j}$ holds if and only if $NQ_{i,j}$ does not contain all NOTIFY messages that $CD_{i,j}$ was waiting for.

Definition 3 (Undesired operation) *Let $Coord_{i,j}$ be the CM generated for the delegate $CD_{i,j}$, and let $NQ_{i,j}$ the notify queue of $CD_{i,j}$. Let $s{\in}S_C$ be the current state reached by $CD_{i,j}$ during its execution. A choreography operation $\alpha{=}r_i.t.r_j$ (for some $r_i, r_j{\in}R_C$ and $t{\in}\mathcal{T}$) is an* undesired operation *in s if and only if one of the following conditions hold:*

- **undesired task:** $\alpha{\notin}Coord_{i,j}(s){\restriction_{Op}}$*;*
- **undesired flow:** $\alpha{\in}Coord_{i,j}(s){\restriction_{Op}} \wedge$ *one of the following conditions hold:*
  - **invalid alternative:** $\exists s^{alt}{\in}S_C^{ALT}, \rho{\in}\Phi : s^{alt} \xrightarrow{\rho}_C s \wedge \neg\rho$*;*

[7] That is, upon receiving a request to perform t, $CD_{i,j}$ was waiting to receive UPDATE_CURRENT_STATE($state$).

- **invalid loop:** $\exists s^{loop} \in S_C^{LOOP}, \rho \in \Phi : s^{loop} \xrightarrow{\rho}_C s \wedge \neg\rho$;
- **missed loop:** $\exists s^{loop} \in S_C^{LOOP}, \rho \in \Phi, s' : s^{loop} \longrightarrow_C s \wedge s^{loop} \xrightarrow{\rho}_C s' \wedge \rho$;
- **missed join:** $\exists s^{join} \in S_C^{JOIN} : s^{join} \longrightarrow_C s \wedge \sim NQ_{i,j}$;
- **deadlock:** $\exists s^{join} \in S_C^{JOIN}, s' : s \xrightarrow{\alpha}_C s' \longrightarrow s^{join} \wedge CD_{i,j}$ *is in* $s' \wedge \sim NQ_{i,j} \wedge \forall (s'', h, k) \in Coord_{i,j}(s)\restriction_{Wait} : CD_{h,k}$ *is in* $s'' \wedge \sim NQ_{h,k}$.

Correctness proof (by contradiction) – Let us suppose that the run-time collaboration of the synthesized CDs and the participants performs an undesired operation. By Definition 3, this means that either an undesired task or an undesired flow has been performed. By referring to the distributed coordination algorithm defined in Section 4.2.2, condition **undesired task** contradicts **Rule 1.1**. Now, let us consider all the cases that characterize condition **undesired flow**. Condition **invalid alternative** contradicts **Rule 1.3.2**; conditions **invalid loop** and **missed loop** contradict **Rule 1.3.3**; condition **missed join** contradicts **Rule 1.3.1.2**; finally, condition **deadlock** contradicts the combined execution of **Rule 1.3.1.1** with **Rule 3** and **Rule 2**. In any case, we deduce a contradiction and, hence, the correctness proof is given meaning that the interaction among the generated CDs and the participants prevents[8] any kind of undesired interaction.

4.2.4. Dealing with Adaptation Issues

By continuing the illustrative example initially introduced in Section 4.1 to clarify the notion of undesired interaction, in this section, we informally discuss an automated adaptor synthesis method that is used in CHOReOS in order to deal with possible adaptation issues. We recall that the latter can arise whenever a service discovered as a participant does not exactly match the role to be played. For a detailed description of the method, we refer to [5, 14]. As already said, the reported adaptor synthesis method has been defined in the context of the CONNECT EU project (`https://www.connect-forever.eu`). The main focus of CONNECT is on the automated synthesis of adaptors/mediators for achieving on-the-fly interoperability of heterogeneous systems. Since the considered heterogeneity is related to adaptation issues that can arise when trying to match the protocol/interfaces of a participant service with the specification of the role to be played, we propose a revision of the method developed in CONNECT in order to make it exploitable in the future for the CHOReOS purposes.

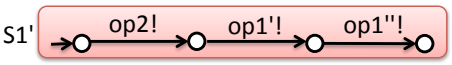


Figure 4.3: $S1'$: a service discovered to play the role of $p1$

Let us suppose that, instead of discovering $S1$, another service, say $S1'$, would have been discovered. The interaction protocol of $S1'$ is shown in Figure 4.3. Let us suppose also that the flow $op1'! \rightarrow op1''!$ (in $S1'$) is semantically equivalent to (yet syntactically different from) $op1!$ (in $S1$). Thus, in order to adapt the protocol of $S1'$ to the one of $p1$, we synthesize an adaptor that reorders the sequence of messages $op2! \rightarrow op1'! \rightarrow op1''!$ into $op1'! \rightarrow op1''! \rightarrow op2!$ and, then, merges the sequence of messages $op1'! \rightarrow op1''!$ into the single message $op1!$. This adaptor is synthesized as a wrapper for $S1'$ with a modular architecture resulting in the concurrent execution of two mediators, one for performing message reordering and the other one for performing the merge of messages. Note that the same adaptor could be used in the case we would consider a service that behaves exactly like $S1$ and its behaviour is changed afterwards so to become the one of $S1'$. This points out the ability of our modular adaptors to solve possible communication and interaction mismatches between the interface/protocol of a participant service and the specification of the role to be played.

[8] As described in Section 4.2.2, in the worst case, an exception is raised.

Choreography modular adaptors synthesis – A modular adaptor is automatically synthesized as a suitable composition of independent *mediators*. A mediator has an input-output behaviour (not necessarily strictly sequential, e.g., for allowing reordering of messages), and it is a "reactive" software entity harmonizing the interaction between heterogeneous services by intercepting output messages from one service and eventually issuing to another service (e.g., a CD) the *co-related* input messages. Message co-relations can be inferred by taking into account ontological information. In particular, we assume the existence of a *Domain Ontology* (DO) that can be used to semantically enrich the protocol description of the considered services[9]. DO represents the relations holding between the various concepts used by the services to be mediated. Typically, ontologies account for two fundamental relations between concepts: *subsumption* and *aggregation*. A concept a is *subsumed by* a concept b, if the set denoted by a is a subset of the set denoted by b. A concept a is *an aggregate of* concepts $b_1, \ldots, b_n$ if the latter are part of the former. It is worth to mention that our use of the ontology concept is specific of the CHOReOS project. Thus, in the following, we will exploit these notions to our purposes. That is, concepts in DO correspond to service input/output operations. The two relations between concepts are, then, used to account for the granularity of the data that define the structure of the messages exchanged by the respective input/output actions. Indeed, in the current practice of ontology development, one cannot expect to find a highly specific (to the considered services) ontology as DO. The production of DO involves the extension of a more general ontology in the application domain. This extension allows the definition of specific ontologies that represent a semantic description for the considered services, respectively. Then DO results from discovering mappings between these ontologies. Note that nowadays there exist several ontologies (e.g., for e-commerce domains, see at: `http://www.heppnetz.de/projects/goodrelations/`) that can serve as common descriptions of specific domains, which can be shared among different applications. Furthermore, they are expressed by using languages (e.g., OWL, DAML, OIL, RDF Schema, just to mention a few) that allow ontology extension and automated reasoning for ontology mapping discovery.

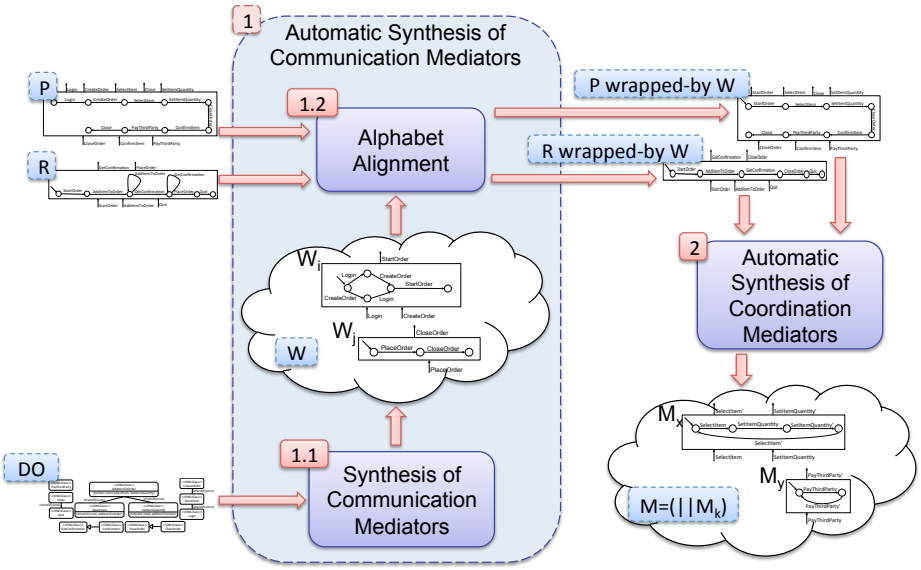


Figure 4.4: Overview of the choreography modular adaptor synthesis

Our modular adaptor synthesis method is organized into two phases. Figure 4.4 pictorially shows the phases (as rounded-corner rectangles) with their related input/output artefacts. The numbers denote the order in which the phases are carried out. The first phase splits into two sub-phases (1.1 and 1.2); it takes as input a domain ontology DO, for services (indeed, for service behavioural descriptions) P and R, and automatically synthesizes a set, W, of *Communication Mediators* (CMs). CMs are responsible for solving *communication mismatches*. They concern the semantics and granularity of the

[9]Concerning the service base, references to semantic information - if any - are specified in the service profile [41]

service protocol actions. To solve these kind of mismatches it is necessary to assume and use ontology knowledge in order to align the two protocols to the same concepts and language. In particular, the CMs in W are used as wrappers for P and R so to "align" their different alphabets to the same alphabet. Roughly speaking, the goal of this phase is to make two heterogeneous service protocols "speak" the same language. To this aim, the synthesized CMs translate an action from an alphabet into a certain sequence of actions from another alphabet (e.g., as illustrated in the previous section, through the merge of messages). However, despite the achieved alphabet alignment, *interaction mismatches* are still possible (e.g., as illustrated in the previous section, some message reordering is needed); the second phase is for solving such mismatches. *Interaction mismatches* concern the control structure of the protocols and can be solved by means of the mediator that can mediate the conversation between the two protocols so that they can actually interact. The synthesis of *Choreography Adaptors* (CAs) is carried out by reasoning on the traces of the "wrapped" P and R. As detailed in [14], for all pairs of traces, if possible, a CA that makes the two traces interoperable is synthesized. The parallel composition of the synthesized CAs represents, under alphabet alignment, the correct modular adaptor for P (e.g., a participant service) and R (e.g., a CD whose coordination logic is based on the specification of the role to be played by P).

4.3. Synthesis in Action

This section describes all the components of the synthesis processor and the REST services that expose them. However, although the Synthesis Processor is a batch process, as part of the overall process supported by the CHOReOS IDRE, we have also implemented a set of Eclipse plugins that, connecting to the REST services, allow developers to transform BPMN2 choreography specifications, to edit, visualize (by using the dedicated GMF-based graphical editor), save, project, simulate CLTSs, etc. The plugins can easily be installed on the ECLIPSE platform by using the dedicated update site: `http://choreos.disim.univaq.it/updatesite/site.xml`. A guide showing the overview of the functionalities offered by the plugins and how to interact with them can be downloaded from the CHOReOS web site (in the dedicated documentation page). Moreover, a report on how the synthesis processor plugins have been applied to the *Airport Arrival Handling* choreography of the *Passenger-friendly Airport* WP6 use case can also be downloaded. The application to the WP6 use case is also reported in Deliverable D1.4(b) [41].

In the following, after describing the REST architecture of the synthesis processor and related REST services, we show how these services have been exploited to automatically derive the CDs for (i) the *In-store Marketing and Sale* choreography of the *Adaptive Customer Relationship Booster* WP7 use case, and (ii) the *Distributed ad-hoc Social Networking* choreography of the *DynaRoute* WP8 use case. While interacting with the REST services, screenshots of the Eclipse plugins will be leveraged to make the presentation more clear.

4.3.1. REST Architecture of the Synthesis Processor

The Choreography Synthesis Processor allows for deriving the CD artefacts from the BPMN2 specification of the choreography. To this end, model transformations are employed and interoperation with the eXtensible Service Discovery (XSD) is required (Deliverable D3.3 [27]). The coordination delegates, when deployed by the Enactment Engine component, allow for enacting the choreography by realizing the distributed coordination logic between the discovered services. The user is supported with bespoke functionalities to:

- start the synthesis process giving as input a BPMN2 Choreography Diagram;
- transform the BPMN2 Choreography Diagram into an extended LTS called choreography LTS (CLTS);

- derive a set of so called Coordination Models containing coordination information that serve to coordinate the services involved in the choreography in a distributed way;
- extract the participants of the choreography and project the choreography on their behavioral role;
- simulate the behavioral role of the participants in the choreography against the behavior of the services discovered by the eXtensible Service Discovery;
- generate the Coordination Delegate artefacts and the so called "ChorSpec" specification to be used by the Enactment Engine component for deploying and enacting the choreography;

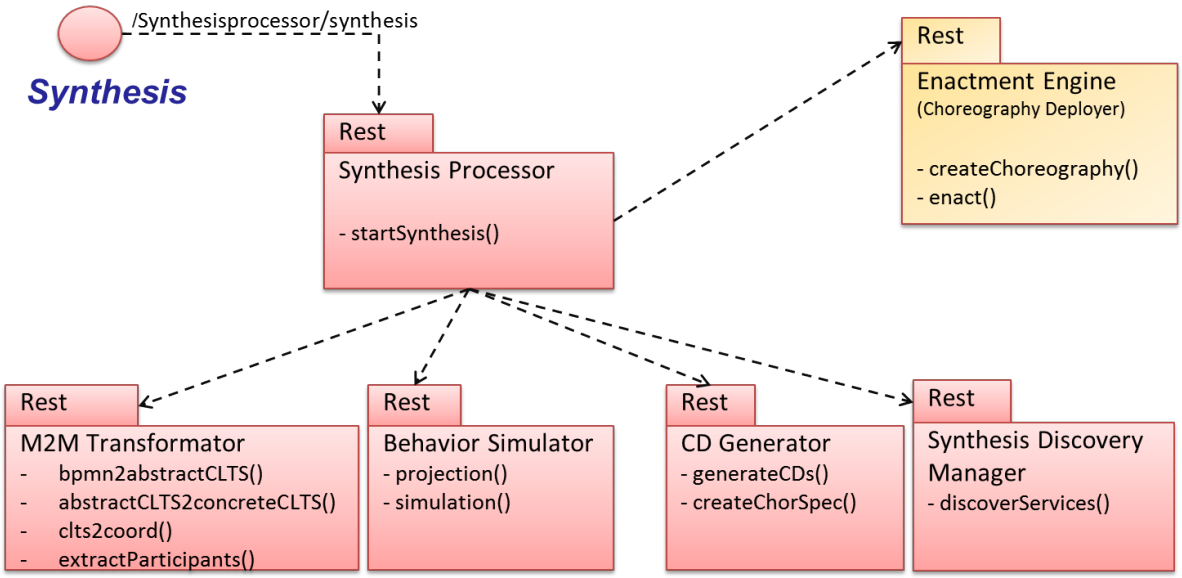


Figure 4.5: REST Architecture of the Synthesis Processor

As shown in the Figure 4.5, the Choreography Synthesis Processor is composed of the components described below. The figure also reports the main functionalities offered by the processor.

M2M Transformator – The Model-to-Model (M2M) Transformator offers a set of model transformations. Specifically, it offers a REST operation `bpmn2clts()` that takes as input the BPMN specification of the choreography and performs the model-to-model transformation to derive the CLTS. The transformation is implemented by means of Atlas Transformation Language (ATL) which is a domain specific language for realizing model-to-model transformations.

Then, starting from the CLTS specification of the choreography, the synthesis process extracts the list of the participants and, applying a further model-to-model transformation, automatically derives, for each participant, the CLTS model of the expected behavior with respect to the specified choreography. To this end another REST operation named `extractParticipants()` is offered. The CLTS model of expected behavior is achieved by projecting (`projection()`) the choreography onto the participant, hence filtering out those transitions, and related states, that do not belong to the participant. Basically, for each participant, this CLTS model specifies the interaction behavior that a candidate service (to be discovered) has to support in order to be able to play the role of the participant in the choreography. Note that, quite big models can be derived when a participant is involved in many tasks of the choreography. This implies to have a considerable number of states and transitions. Many of these states might be forking and join states (also nested). This makes the CLTS treatment difficult, if not impossible, without being supported by a synthesis approach that is able to automatically calculate all the possible operations interleaving when interacting with other participants.

Synthesis Discovery Manager – The Synthesis process and the Discovery process interact each other to retrieve, from the service base, those candidate services that are suitable for playing the participant roles required by the choreography specification, and hence, those services whose (offered and required) operations and behavior are compatible with the expected behavior as extracted from the choreography through projection. In particular, for each participant, the Synthesis Processor interacts with the Synthesis Discovery Manager by performing the REST call to the `discoverServices()` operation that takes the participant (abstract) CLTS as input. Then, a REST query is issued to the eXtensible Service Discovery (XSD) component (Section 3.2).

Behavior Simulator – Once a set of concrete candidate services has been discovered (see the Synthesis Discovery Manager component), the synthesis process has to select them by checking, for each participant, if its expected behavior can be simulated by some candidate service. Note that, for a given participant, behavioral simulation is required since, although the discovered candidate services for it are able to offer and require (at least) the operations needed to play the role of the participant, one cannot be sure that the candidate services are able to support the operations flow as expected by the choreography. Thus, in order to simulate the expected behavior of a participant with the behavior of a service, the Behavior Simulator offers a REST operation named `simulation()` that takes as input the projected (abstract) CLTS of the participant and the extended (concrete) LTS of the service as retrieved by the URI returned by the discovery service. It might be interesting to mention that the simulation method implements a notion of strong simulation suitably extended to treat the CLTSs and extended the LTSs we use in CHOReOS. After simulation, if all the participant roles have been "covered" by (some of) the discovered services, the abstract CLTS is concretized with the actual names of the selected services and the actual names of the offered and requested operations. This step is performed by the REST operation `absctractCLTS2concreteCLTS()`. Then, the automated synthesis process distributes the coordination logic specified by the obtained CLTS into a set of Coordination Models by means of the functionality `clts2coord()`. The simulation features of the Behavior Simulator are also offered by the Eclipse plugin we have aptly developed. Thus, considering the CLTS of a participant and the extended LTS of one service that has been discovered as a candidate for it, the developer can easily use the plugin to check if the CLTS is simulated or not.

Coordination Delegate Generator – Once the services have been selected for all the choreography participants, and hence the CLTS has been concretized, the synthesis processor can generate the Coordination Delegates through the REST operation `generateCD()` offered by the Coordination Delegate Generator component.

Next step in the process – Once the Coordination Delegates have been generated, the Coordination Delegate Generator component can further generate a specification of the choreography (called `ChorSpec`) to be passed to the Enactment Engine. To this end, the REST operation `createChorSpec()` is offered. It takes as input the selected services and the coordination delegates generated for them. The `ChorSpec` is an XML-based declarative description of the choreography that specifies the locations of the selected services and of the generated Coordination Delegate artifacts that can be deployed. Indeed, before passing the `ChorSpec` to the Enactment Engine, the Choreography Offline Testing process activity is performed to asses the quality of the choreography specification, its well formedness, etc.

== IMPLEMENTATION == The implementation of the synthesis processor, as well as examples, can be accessed from `http://websvn.ow2.org/` under the trunk `choreos/trunk/choreography-development/synthesis/rest/`.

== REQUIREMENTS == The main requirement for the Choreography Synthesis Processor is to have a server to install and run the processor as a service. Such server may be able to run Java 6 and Maven 3.

== CONFIGURING IT == The components of the synthesis processor do not require any particular configuration. However, for each component the file "configuration.properties" can be edited in order to change, e.g., the server port, server name etc. For instance, for a local machine deployment, the configuration.properties file in `choreos/trunk/choreography-development/synthesis/rest/synthesis-processor/src/main/resources/configuration.properties` looks like the following text:

SERVER_NAME = `Synthesis Processor`
SERVER_PORT = `10000`
BEHAVIOUR_SIMULATOR_HOST = `http://localhost:10002/behavioursimulator`
CD_GENERATOR_HOST = `http://localhost:10003/cdgenerator`
M2M_TRANSFORMATOR_HOST = `http://localhost:10001/m2mtransformator`
SERVICE_DISCOVERY_HOST = `http://localhost:10004/servicediscovery`
CHOREOGRAPHY_DEPLOYER_HOST = `http://localhost:9102/enactmentengine`

== COMPILING IT ==

After installing Maven 3, it is enough to compile the Choreography Synthesis Processor distribution using this command:

```
mvn clean install
```

== RUNNING IT ==

For all the above components, after the compilation step, go to the project home (see Table 3.2) and use this command to run the component:

```
mvn exec:java
```

Note that the enactment engine must be started as well (see the Enactment Engine installation guide).

== VERIFYING IF IT WORKS ==

Once started successfully, in the console a message will appear. For example:

```
INFO i.u.d.c.synthesis.rest.RESTServer - Synthesis Processor has started
[http://localhost:10000/synthesisprocessor/]
```

4.3.2. Eclipse Plugins of the Synthesis Processor

The synthesis processor is a fully automated process. That is, given a choreography specification as input (passing through all the model transformations and the intermediate models manipulated my them) a set of CD artefacts and the corresponding final `ChorSpec` is directly produced as final output. Thus, in this section, we briefly describe one scenario of the WP7 use case and one scenario of the WP8 use case. Then, in the Appendixes A.1 and A.2 a set of screenshots of the Synthesis Eclipse plugins will be leveraged to show how the REST services interact each other, in a step-by-step fashion, while being applied to the described use cases. Indeed, as it will be clear in the appendix, the Synthesis plugins allow the user to choose if to execute whether the Automatic Synthesis Processor that (calling in sequence all the Synthesis REST services) produces all the artefacts in one step, or the Interactive Synthesis Processor that produces the artefacts step by step.

According to the CHOReOS Development Process defined in Deliverable D2.2 [36], in the appendixes, we give evidence of the fact that (i) each single feature previously designed for the synthesis processor has been implemented by a dedicated REST service, and that (ii) the overall synthesis process flow has been implemented and fully integrated into the CHOReOS IDRE. The evaluation of the application of the Synthesis Processor to the WP7 use case and to the WP8 use case is reported into the Deliverable D7.5 [38] and the Deliverable D8.4 [39], respectively.

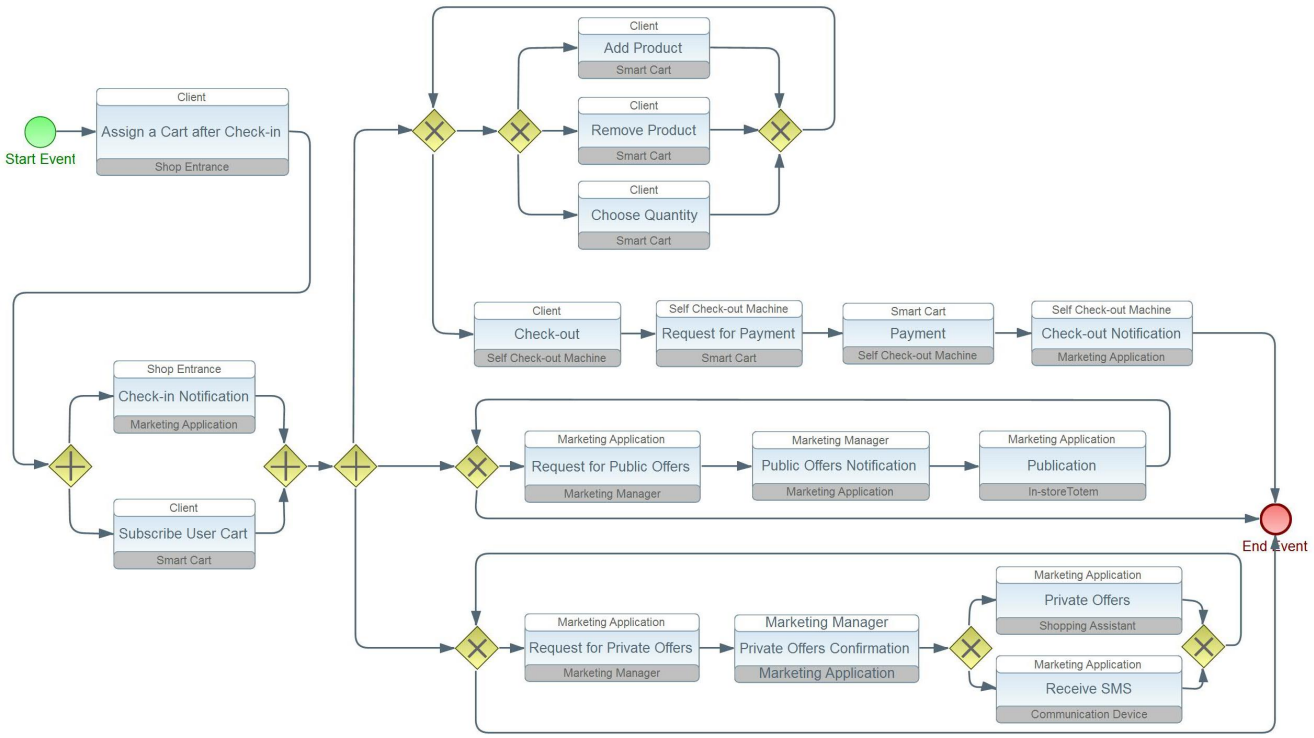


Figure 4.6: In-store Marketing and Sale choreography - WP7

In-store Marketing and Sale choreography - Adaptive Customer Relationship Booster (WP7)

The *In-store Marketing and Sale* choreography shown in Figure 4.6 is the part of the Adaptive Customer Relationship Booster systems which aims to monitor the activity of a client inside the shop in order to propose him/her tailored shopping offers and/or advertisements according to the user information (preferences, current shopping list, etc.) held by the shopping assistant application service. Moreover, within this choreography, the marketing application service manages, in cooperation with the marketing manager service, the public products offer inside the store using the in store totem service. Note that, the choreography diagram in the figure does not graphically show the input messages and output messages of the choreography tasks, which are however internally specified by using the XML schema (which is the default language for specifying BPMN2 `Item Definition` elements, and hence messages[10]). The specification of the messages is then used by the Service Base Query Engine (see Section 3.2.1) to discover suitable services (see Appendix A.1).

The choreography is triggered by the `Client` entering the shop. The `Shop Entrance` service detects the presence of a specific `Client` inside the store, assigning him a cart and notifying the `Marketing Application` about the new `Client`. Once the `Client` has subscribed to his virtual cart, he can perform shopping by adding and removing products from it. While `Client`s are inside the shop, both targeted and public marketing activities are performed. The `Marketing Application` keeps asking the `Marketing Manager` for new public offers and advertisements, which are then pushed by the `Marketing Application` itself onto the `In-store Totem`s. Private offers and advertisements are also constantly requested by the `Marketing Application` to the `Marketing Manager`, and are delivered to the `Client`s inside the shop by the `SMS` service and by sending them to their `Shopping Assistant` Apps loaded inside their smartphones. Once the `Client` finishes his shopping activities, he/r goes to a `Self Check-out Machine` to perform payment. This is achieved with the interaction between the `Self Check-Out` and the `Smart Cart` services. This choreography contains many parallel (see the parallel branches represented as an rhombus marked with a “+” with two outgoing arrows and the merging branches represented as an rhombus marked with a “+”

[10] `http://www.omg.org/spec/BPMN/2.0/`

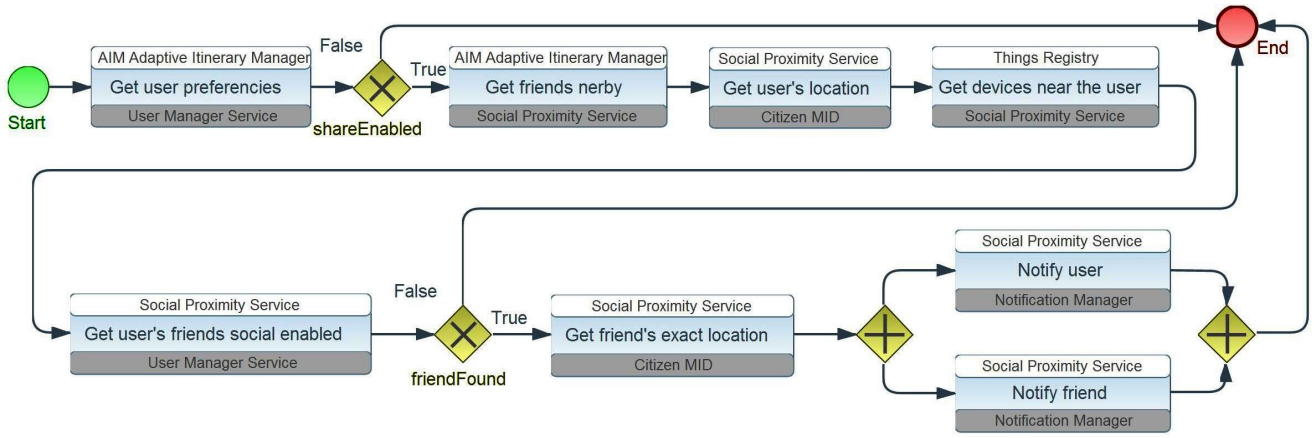


Figure 4.7: Distributed ad-hoc Social Networking choreography - WP8

with two incoming arrows) and alternative flows (see the alternative exclusive branches represented as an rhombus marked with a "x" with two outgoing arrows) that must be properly coordinated. This scenario contains a number of forking branches that must be joined at later execution points, as well as conditional branches, which make it interesting.

Distributed ad-hoc Social Networking choreography - DynaRoute (WP8)

The choreography in Figure 4.7 models a *Distributed ad-hoc Social Networking* scenario, which concerns the collaboration of location-based services and citizen mobile apps. The choreography considers user preferences, user friend lists, and user location under predefined private data policy, to create ad-hoc itineraries between two citizens that know each other and are willing to meet in some place. In brief, upon receiving a start event from a citizen (i.e., from the dedicated mobile app named `Citizen MID`), the `AIM Adaptive Itinerary Manager` (`AIM`) interact with the `User Manager Service` (`UMS`) to check if location sharing is enabled according to the citizen preferences. If yes (see the alternative exclusive branches represented as a rhombus marked with a "x"), `AIM` starts matching the GPS position of all citizen contacts by using the `Social Proximity Service` (`SPS`). In sequence, `SPS` gets the location of the user, interacts with the `Things Registry` to get the list of all devices found nearby, and then checks for a friend with location sharing is enabled. If a friend is found, `SPS`s gets the exact location of the friend. Finally, after the friend has been selected by the requesting citizen, and the friend has accepted, `SPS` starts two concurrent threads to notify in parallel the `Notification Manager` (`NM`) for both the requesting citizen and the friend (see the parallel branches represented as an rhombus marked with a "+" with two outgoing arrows). The started threads are joined afterwards as soon as both the notification tasks have been accomplished (see the merging branches represented as an rhombus marked with a "+" with two incoming arrows). An important aspect of this simple scenario is that the two friends can undertake the itineraries created ad-hoc for them only after being both notified.

5 Analysis Tools

In this chapter, we present the two tools we developed in order to enable the analysis of choreographies: the QoS analyzer and the stability and interdependencies analyzer (dependency analyzer, for short)

5.1. QoS Analyzer

The objective of the QoS analysis is to examine the non functional properties of choreographies of services. The result of the analysis can serve at the runtime and the design-time. In the former, the projection of the *non-functional behavior* of a choreography is used for deciding on the appropriate manner of adapting its execution. In the design-time, the analysis provides recommendations to the end-user about the best configuration (services, machines etc.) to use in running the choreography.

5.1.1. QoS Analyzer at a Glance

The non-functional properties (or QoS dimensions) that are manipulated in the QoS analysis components are: the service response time, the availability, the reputation, the energy consumption, the price. The components of the analysis manipulate these properties through two type of representations: the deterministic and the probabilistic ones.

Deterministic representations

The QoS dimension of each service operation in these cases is associated with a real scalar value or a function of the operation input' size. In the case where the QoS are modeled by the mean of functions, the analysis is made per request. Each request is defined by an initial input provided for starting the run of the choreography. In the case where scalar values are used, the analysis is representative of the behavior of a class of the class of requests for which the mean QoS of operations are the ones that are used. When using scalar representations, the behavior of the services' choreographies is analyzed in three cases: the best behavior for the optimization of a QoS dimension, the mean behavior that traduces what can in general be expected and the worst possible behavior on a QoS dimension.

Probabilistic representation

Each QoS dimension of a service' operation is associated in these cases with a tuple of the form (X, f_X). Here, X is the finite set of possible values for the QoS values and f_X is the probability mass function ruling the occurrence of these values. For instance, the response time can be modeled with the valued $X = \{0.4, 5, 8.2, 12\}$ (in ms) and $f_X = \{0.2, 0.3, 0.3, 0.4\}$. This will mean that the response time will be equal to 0.4ms with the probability 0.2, 5ms with the probability 0.3, 8.3ms with the same probability and 12ms with the probability 0.4. The probabilistic representations are important when there is an important potential variation in services performances. They are well suited for capturing the behavior of services on clouds architectures.

5.1.2. QoS Analyzer Revised

The QoS analysis of choreographies in the project is realized by three key components: the QoS predictor, the QoS recommender, and the scalability predictor.

The QoS predictor

The QoS predictor is the core component of the analysis. Given a choreography of services whose operations behavior is specified, the predictor computes the results that can be expected on each QoS dimension. More formally, the functioning of the predictor can be summarized as follows.

INPUT: A choreography of services described with the BPMN collaboration notation, the Qos values of the services' operations for each dimension (service response time, energy consumption, availability etc.)

OUTPUT: The QoS expected from the choreography on each dimension.

	Deterministic	**Probabilistic**
Structured comp.	Graph reduction	Graph reduction
Unstructured comp.	Linear programming	

Table 5.1: QoS prediction approaches

The QoS predictor ranges the services' choreographies in two classes, depending on the *regularity* of their structure: structured and unstructured compositions. On the first class, it bases its prediction on the graph reduction approach; in the second class, it uses a linear programming based approach.

We provide in Figure 5.1, an illustration of the graph reduction approach. Initially, the BPMN choreography is given by the graph of Figure 5.1(1). The first stage of the reduction consists of computing a reduction order. The order decomposes the choreography in elementary subgraphs [12]. Then, one retrieves the top element in the order (it refers to a subgraph) and reduce it. This reduction consists of replacing the subgraph by a single node, aggregating its QoS. This gives us the graph of Figure 5.1(2). One continues in the same way until the reduction order is empty. At this stage, the initial BPMN choreograhy will be reduced to a unique node. Its QoS values will then be returned.

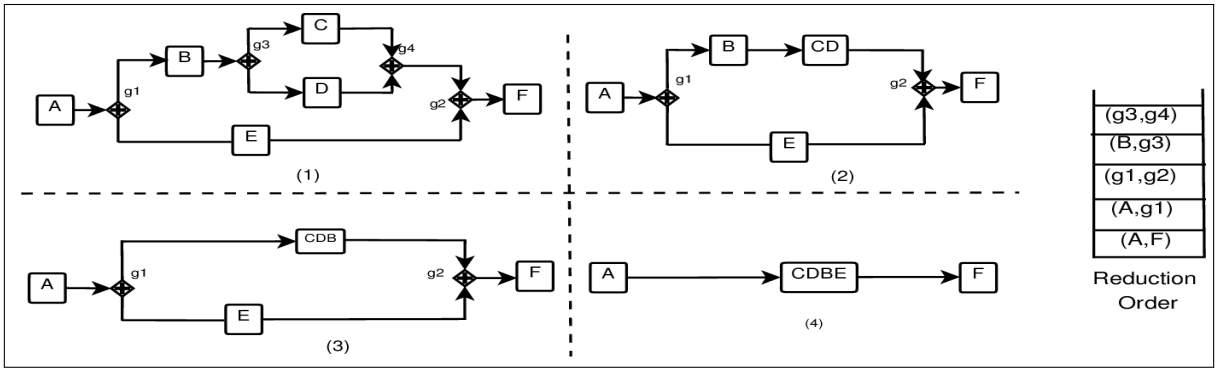


Figure 5.1: Example of graph reduction.

Given a BPMN graph representation, the prediction based on linear programming comprises the stages described in the Figure 5.2. Though the QoS predictor will consider the composition of Figure 5.1 as structured, the linear programming approach can be applied on it. For the service response time, the equations of Figure 5.3 will be generated in the second stage of the linear programming approach.

After the model resolution, tF will comprise the mean service response time. The QoS predictor uses the GLPK solver[1] for computing this value.

[1] http://www.gnu.org/software/glpk/

1. **Creation of a variable for each operation and QoS dimension;**
2. **Generation of Linear model composed of equations on these variables;**
3. **Resolution of the linear model.**
4. **Extraction of the results.**

Figure 5.2: Stages of the prediction with linear programming

1. $tA \geq 0$	7. $tG1 \geq 0$
2. $tB \geq 0$	8. $tG2 \geq 0$
3. $tC \geq 0$	9. $tG3 \geq 0$
4. $tD \geq 0$	10. $tG4 \geq 0$
5. $tE \geq 0$	11. $tG5 \geq 0$
6. $tF \geq 0$	
12. $tG1 \geq T(A)$	16. $tC \geq tG3 + T(G3)$
13. $tE \geq tG1 + T(G1)$	17. $tD \geq tG3 + T(G3)$
14. $tB \geq tG1 + T(G1)$	18. $tG4 \geq tC + T(C)$
15. $tG3 \geq tB + T(B)$	19. $tG4 \geq tD + T(D)$
20. $tG2 \geq tG4 + T(G4)$	21. $tG2 \geq tE + T(E)$
22. $tF \geq tG2 + T(G2) + T(F)$	

Figure 5.3: Equations of the response time

The QoS recommender

The prediction component can be interrogated throughout a recommender for finding the best configuration to adopt for running a choreography. This is necessary for instance, for choosing the best services to use in the processing of a choreography with some SLAs constraints. In the recommender, this problem is formulated as a combinatorial one whose objective is to minimize a penalty function. The typical use cases of the recommender can be summarized as follows.

INPUT: A choreography of abstract services described with the BPMN collaboration notation, the set of possible concrete services to associate with abstract ones and their QoS values

OUTPUT: Best association between abstract and concrete services, the penalty value of this choice.

We did multiple evaluations of the recommender system. An example is provided in Figure 5.4. Here, we compared in the airport transportation choreography, the recommender with two local optimization based-approaches. We considered various settings that differ on the maximal number of services used, for the run of each operation. In the example, the penalty are estimated on the service response time and the energy consumption. As one can notice, the recommender found in each cases, the solution of lower penalty. Additional details about the comparison setting can be found in [18].

The Scalability Predictor

As the QoS recommender, this component depends on the QoS predictor. The scalability prediction is made per QoS dimension. In each, the component makes scale the input configuration of choreographies (in a *windows of time or observation*) for generating a synthetic report of the estimations computed by the QoS predictor. More formally, the typical functioning of the scalability predictor can be summarized as follows.

INPUT: A choreography of services described with the BPMN collaboration notation, the period of time during which the scalability is evaluated, the local QoS values in each dimension during this windows of time, the Qos values of the services' operations

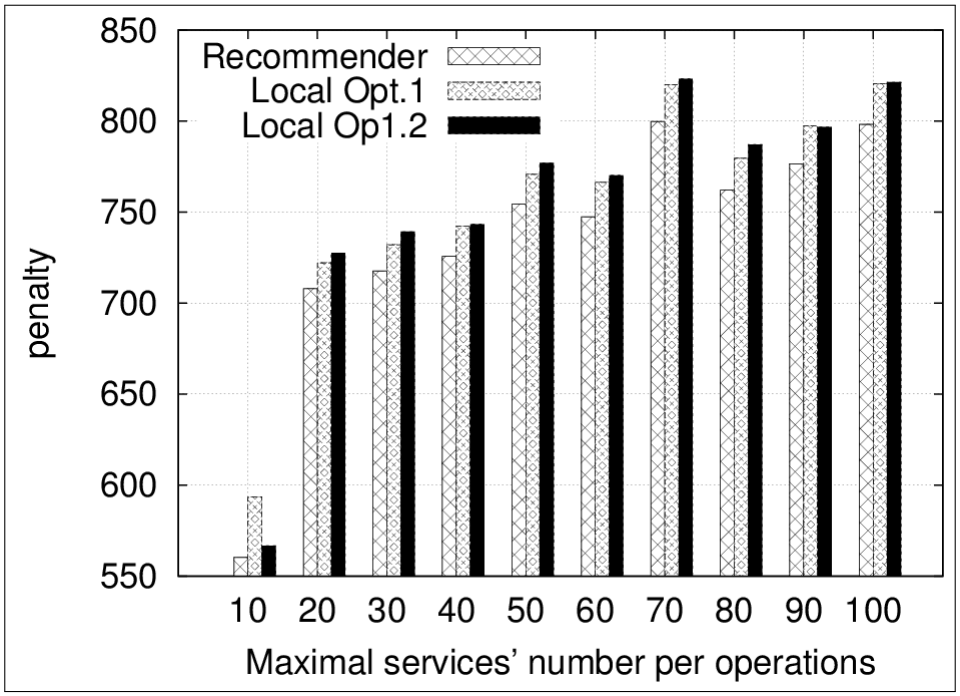


Figure 5.4: Mean penalties on the service response time and energy consumption

OUTPUT: The QoS expected from the choreography on each dimension and during the windows of observation.

5.1.3. QoS Analyzer in Action

The CHOReOS project is interested in optimizing the collaboration among various processes collaborating on airport transportation (WP6). A special collaboration use case is the crisis management when facing bad weather condition during a flight. This collaboration involves an air traffic control (ATC), a pilot (more precisely an internal plane system), an airport and an airline. We can describe the collaboration as follows. Due to bad weather conditions, the ATC decides to cancel all flights. It checks for scheduled flights, reroutes them and notifies all concerned pilots. The pilots inform their airlines and passengers. The ATC also informs about its decision the airport at which the flight is rerouted. This airport then prepares its ground staff for the new situation.

A complete description of the use case is provided in D6.1 [34] and D6.2 [31]. In our work, we consider only the sub-view presented in Figure 5.5.

There are multiple possible Web implementations of the process given in Figure 5.5. In particular, we can either use a sub-composition for implementing an activity or a WS operation. In our experiment we will consider the latter option for all activities.

Experimental setting

In the crisis management example, we performed 10 series of 1000 experiments. These experiments had two objectives: the first was to show that the proposed MILP (Mixed Linear Integer Program) approach has a real qualitative gain over local optimization. The second was to show that although we use MILP, reasonnable time can be expected in practice.

Each series is determined by the parameters $MaxE$ and $MaxS$ (respectively, related to energy and response time) used for SLAs (Service Level Agreements). The ones that we used are set in Table 5.2. For each experiment, we created multiple concrete operations to be associated with abstract ones. The number of concrete operations belongs to the set $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. For any

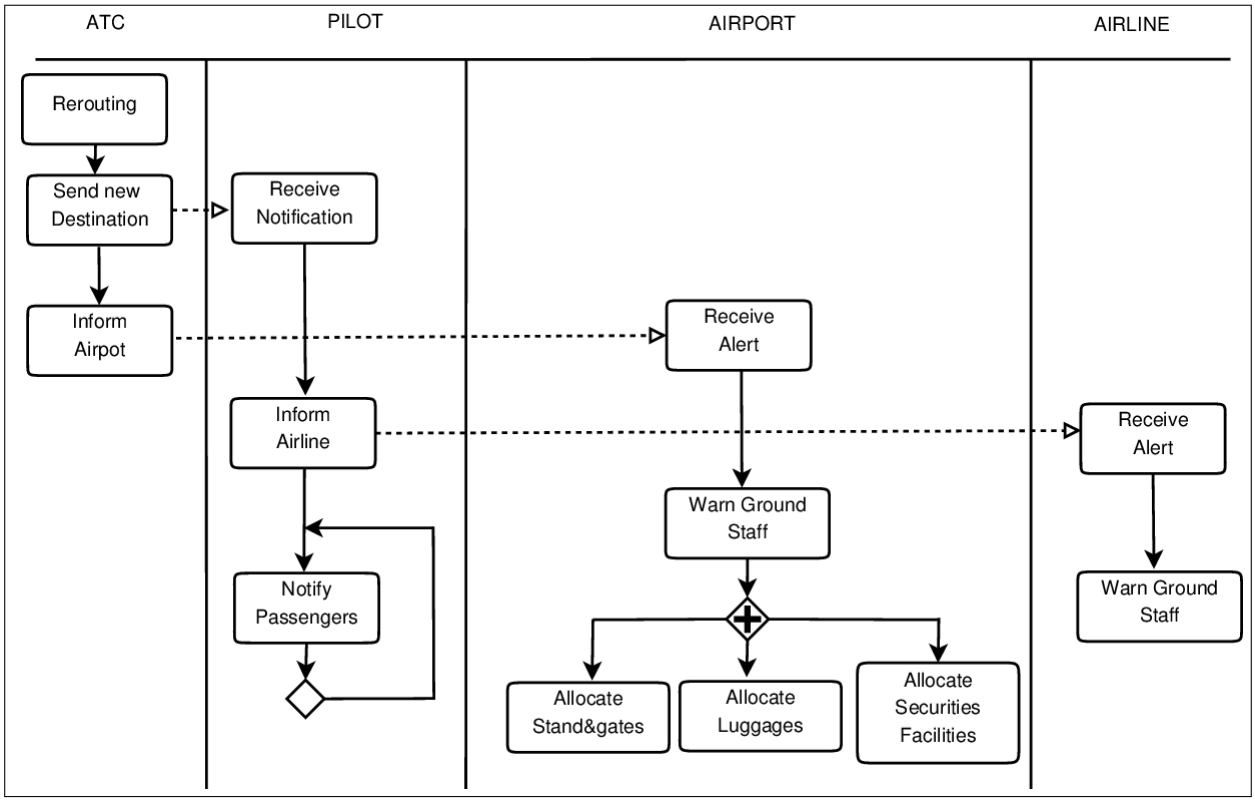


Figure 5.5: Crisis management in airport transportation

experiment, we always chose the same number of concrete services for all abstract operations.

The concrete operations SRT (Service Response Time) values are drawn from the uniform distribution within $[100, 1000]$ (in ms). Assuming that an operation has an SRT equal to S and leads to a mean power consumption equal to P during its execution, we used the formula $E = P.S$ for computing its energy consumption. We draw P (in watts) from the uniform distribution in the interval $[100, 200]$. Finally, we chose the number of loops for passenger notification from the uniform distribution between 1 and 10, the communication times between 10 and 40 (ms) and we normalized E in order to have a metric in watt-second.

We performed our experiments on an Intel Core i7 processor, 2.7 Ghz, 8 Gb. The implementations were done in C language with the GLPK solver[2].

#Series	1	2	3	4	5	6	7	8	9	10
$MaxE$	1600	1600	1700	1700	1800	1800	1900	1900	2000	2000
$MaxS$	1600	1650	1650	1700	1700	1750	1750	1800	1800	1850

Table 5.2: Experimental settings. Given a serie number (as e.g. 1) and a maximal number of concrete services (as e.g. 20), we performed 100 experiments. We did not change the value of w in the experiments. We always have $w = 0.5$. With $w = 0.5$ we have the same priority for SRT and Energy.

Experimental results

In Fig. 5.6, we show the mean aggregated penalties obtained from our experiments with our MILP approach and the *EC_Min_First* and *SRT_Min_First* policies. The aggregates are made on the number

[2]http://www.gnu.org/software/glpk/

of concrete operations used in the experiment. These results are the optimal ones that the GLPK solver computed in each case. The results describe an increase of the penalty on the maximal number of concrete operations. This is due to the fact that in increasing the number of concrete services, we increase the diversity (or the standard deviation) in SRT and EC per operations. We also did a comparison of the penalties between the local optimization policies and MILP. The results are reported in Table 5.3 and confirm the superiority of MILP over local optimization as suggested by our previous analysis.

For the sake of fairness, we aggregated the penalties values in the experiments only when there were no approaches leading to an infeasible solution. In many cases indeed, the given problems were infeasible. Figure 5.7 reports for each approach the number of infeasible solutions that were detected. These results globally confirm the need to establish a negotiation procedure with the users for preventing infeasible problems. As shown by the results, it does not suffice to increase the maximal service response time or the energy consumption to have less infeasible cases. This number depends on the combination of these two parameters but also on the QoS of the concrete services. For concluding on infeasibility, we compared MILP and local optimization. The results, presented in the table 5.7, state that in all cases where a solution was found by local optimization, we found one with the MILP.

Finally in Table 5.5, we present the MILP runtimes. The results are encouraging they show that despite the NP-hardness of the service selection problem, we can expect to solve it in a reasonnable runtime even with more than 1300 concrete operations. Let us however notice that we do not expect such results if we increase both the number of abstract and concrete services.

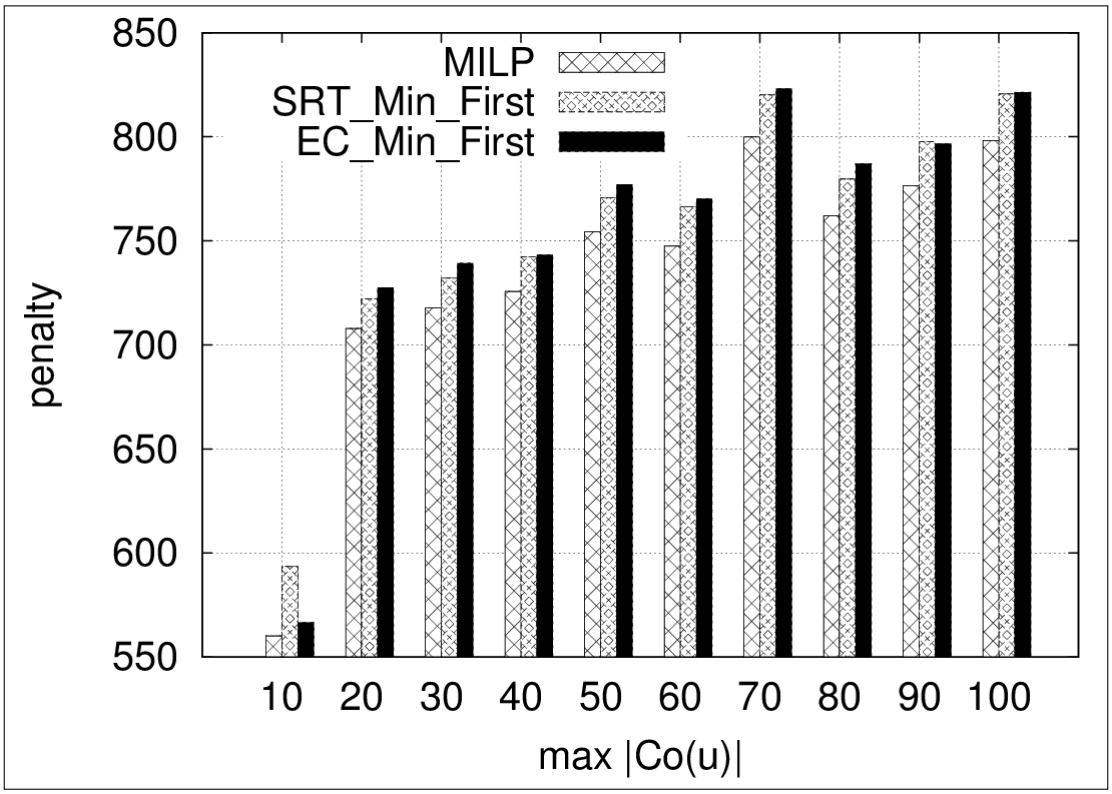


Figure 5.6: Mean penalty per approaches depending on the maximal number of concrete operations

Limitations

To run the previous experiments we relied on synthetic values which initially appeared to be reasonable. From them we were able to find feasible solutions for some instances. A similar approach has to be conducted when dealing with realistic values.

$\max_{u \in O} \lvert Co(u) \rvert$	SRT_Min_First	EC_Min_First
10	1.059	1.011
20	1.020	1.027
30	1.022	1.030
40	1.021	1.024
50	1.025	1.029
60	1.025	1.030
70	1.022	1.029
80	1.026	1.032
90	1.027	1.025
100	1.028	1.029

Table 5.3: Ratio of mean aggregated penalties between the local approaches and the MILP

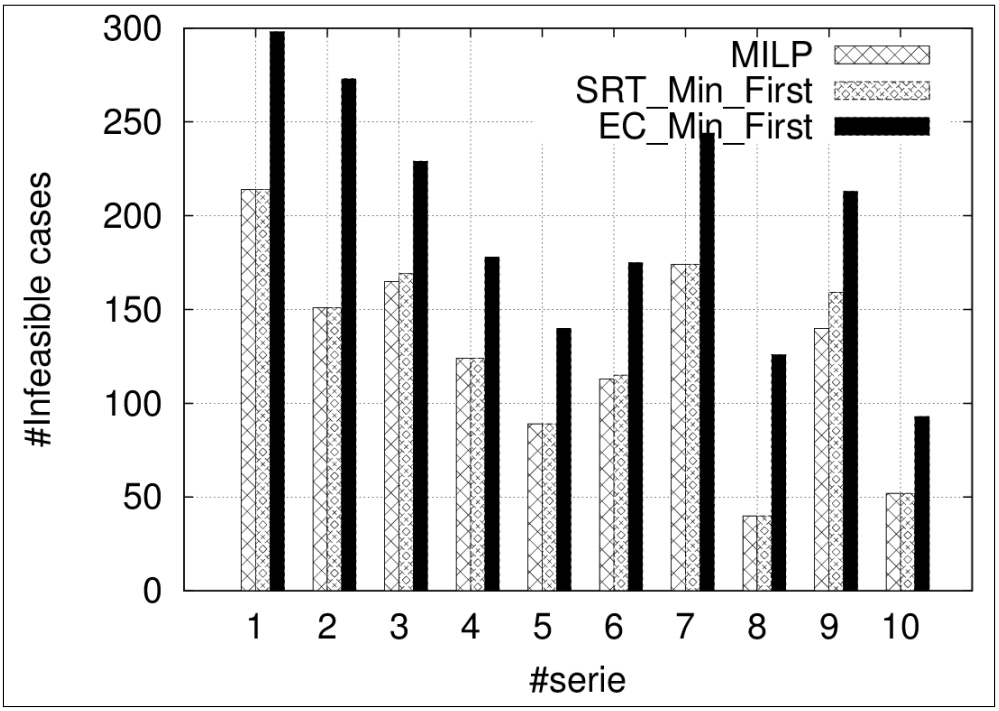


Figure 5.7: Number of infeasible cases per approaches

$\#serie$	SRT_Min_First	EC_Min_First
1	1.000	1.392
2	1.000	1.807
3	1.024	1.387
4	1.000	1.435
5	1.000	1.573
6	1.017	1.548
7	1.000	1.402
8	1.000	3.15
9	1.136	1.521
10	1.000	1.788

Table 5.4: Ratio of the number of infeasible cases between the local approaches and the MILP

#Serie	1	2	3	4	5	6	7	8	9	10
Runtime	5.0	5.2	6.3	6.8	6.7	8.0	10.5	8.6	9.1	8.9

Table 5.5: MILP Runtime (in 10^{-2} seconds)

5.2. Dependency Analyzer

In this chapter, we throughly describe the dependency analyzer tool and its main features. This chapter is structured as follows: In Section 5.2.1, we present the main goals of the dependency analyzer and provide a brief overview of how it works. In Section 5.2.2, we present its features and the kind of questions it answers. Finally, in Section 5.2.3, we run the tool on some choreography diagrams from CHOReOS use cases and present the results we obtained.

5.2.1. Dependency Analyzer at a Glance

Evolving and maintaining choreographies requires caution, since inappropriate changes may induce side and/or ripple effects. A side effect is an error or other undesirable behavior that occurs as a result of a modification. In turn, a ripple effect is the effect caused by making a small change to a system which affects many other parts of a system. Given this scenario, mechanisms to perform change impact analysis [3] become crucial. Most part of change impact analysis concerns making the existing relationships among artifacts more explicit to humans, so that they can maintain and evolve software systems more easily. Most importantly, change impact analysis enables one to identify the potential effects of a change and to estimate what needs to be modified to accomplish a change, thus helping prevent side effects and estimate ripple effects. The dependency analyzer is a change impact analysis tool developed for the specific context of CHOReOS.

The initial task of the dependency analyzer concerns capturing dependencies. In summary, the analyzer traverses the structure of choreographies and detects dependencies among participants by discovering who communicates with whom. In the same sense, dependencies among choreographies are also detected, since they may invoke each other. Finally, dependencies from choreographies to services are also detected, since choreographies are realized and implemented with concrete services.

After dependencies are captured, the dependency analyzer builds graphs and computes stability and change impact by relying on social network analysis techniques [46] and fundamental graph algorithms [17]. Furthermore, since choreographies can be large, complex, and depend on one another, inspecting them in isolation becomes infeasible. To this purpose, the dependency analyzer also relies on both low-level and high-level visualization techniques to help humans understand and reason about change impact.

5.2.2. Dependency Analyzer Extensions

The dependency analyzer is applicable in two different moments of the CHOReOS development process: before service binding and after service binding. For each of these moments, the tool offers a set of features that aim to help choreography designers and developers maintain and evolve their choreographies. In particular, such features are driven by questions that often arise during change impact analysis activities. In the following, we list the questions we focused on and how the tool addresses each of them.

Before Service Binding

After going through requirements, choreographies are specified in BPMN2 diagrams. In this context, our tool supports two kinds of analysis: intra-choreography analysis and *inter-choreography analysis*.

We now focus on the former, which restricts the analysis scope to a single choreography (and its subchoreographies). The tool aims to answer the following change impact questions:

Q1) *If I removed a certain participant P_i from a choreography C, which other participants (from the same choreography) would be directly impacted?*

The answer says which other participants would be impacted. For instance, it might be the case that a certain choreography has 10 participants and one of them becomes irrelevant to the business scenario covered by the choreography. Answering this question informs which participants would be affected (e.g., would have to change the code of their web services that realize the choreography in question).

In order to answer this question, the tool relies on the *participants call-graph* from choreography C. In this directed graph, vertices denote participants and edges denote calling relationships between pairs of participants. For instance, if a participant P_i invokes an operation from participant P_j, then we create vertices P_i and P_j, and draw an edge from P_i towards P_j. We highlight that we are employing here the same convention used by the synthesis processor: the name of a choreography task denotes an operation provided by all non-initiating (target) participants in that task (see Figure 4.2). We also enrich the graph's edges with the name of the invoked operations. More precisely, each edge has a label that stores the name of each operation invoked by the initiating participant. We build this graph using the following algorithm (described in pseudo-code):

Algorithm *calcParticipantsCallGraph(c)*
(∗ Given in a certain choreography c, this algorithm builds the participants call-graph. Such graph denotes how participants c

Input: Choreography c
Output: Participants call graph g

```
1.  g ←new DirectedGraph()
2.  q ←new Queue()
3.  q.add(c)
4.  while (q is not empty)
5.     do c ←q.poll()
6.        for each choreography task t in c
7.           do ip ←t.getInitiatingParticipant()
8.              tps ←t.getTargetParticipants()
9.              if (g does not contain vertex ip)
10.                then g.addVertex(ip)
11.             for each tp in tps
12.                do if (g does not contain vertex tp)
13.                      then g.addVertex(tp)
14.                   if (g does not contain edge (ip,tp))
15.                      then g.addEdge(ip,tp)
16.                   edge ←g.getEdge(sender,receiver)
17.                   edge.addOperation(t.getName())
18.       for each subchoreography sub in c
19.          do q.add(sub)
20. return g
```

In a nutshell, we traverse the choreography (and its subchoreographies (5.8)) capturing all operation invocations and updating the call-graph accordingly. Hereafter, we reference to this call-graph as M_1. We answer Q1 by determining the direct predecessors of P_i in M_1, that is, the participants that directly depend on P_1.

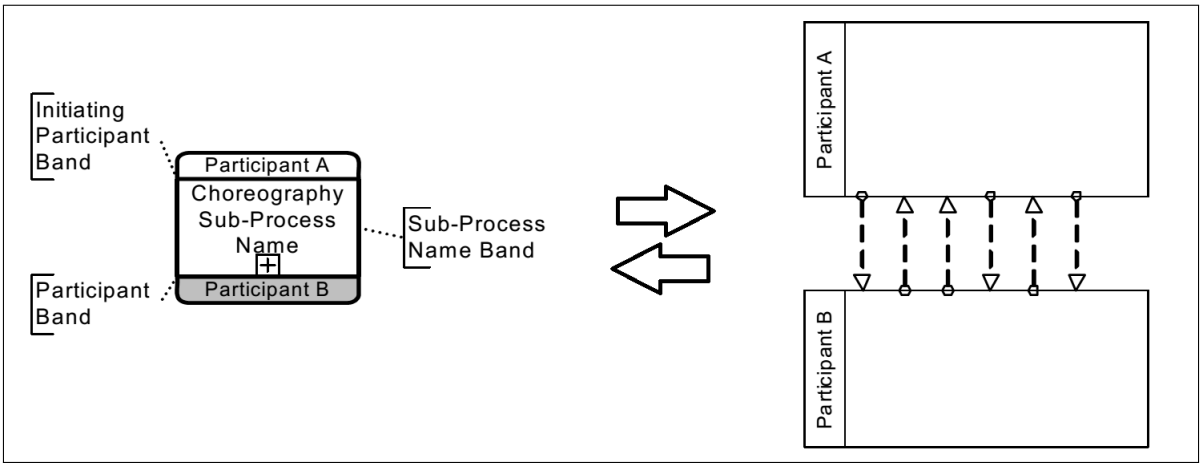


Figure 5.8: BPMN 2.0 subchoreographies

Q2) *If I changed the signature of or removed an operation o_i from participant P_j from choreography C, then which other participants (from the same choreography) would be directly impacted?*

It is natural to suppose that participants will eventually have their provided interface changed due to business needs. For instance, a certain participant might change the signature of a certain provided operation (e.g., one more input parameter is added). By analyzing choreography BPMN2 diagrams, it is possible to determine the sets of required and provided interfaces of all included participants.

Similarly to the previous question, we address Q2 by also building M_1 and inspecting the direct predecessors of P_j. However, we limit the set of predecessors by selecting only those that invoke the operation o_i from P_j. We achieve this by simply inspecting the labels from the incoming edges of P_j in M_1.

Q3) *If I removed a certain participant p from a choreography c, then what would be the impact on cC?*
gskip

This question is similar to Q1. However, in this case, the focus of the impact analysis is on the choreography itself (and its subchoreographies). For instance, it might be the case that business analysts would like to evaluate the impact of removing a certain participant from a choreography. This sets a new challenge to our analysis, since now the structure of the choreography must be carefully taken into account and analyzed.

We tackle this question by adapting workflow analysis algorithms we developed in past studies [20]. The algorithm *calcImpactOfRemovingParticipantFromChoreography(p,c)* takes a participant p and choreography c as input and determines the impact of removing p from c.

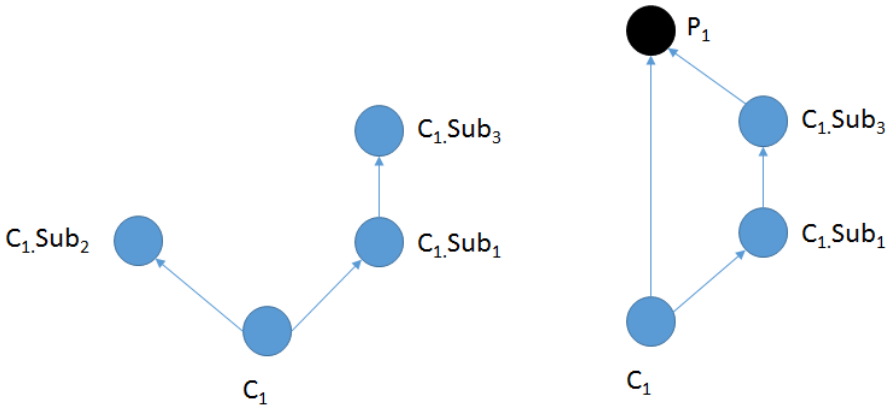
Algorithm *calcImpactOfRemovingParticipantFromChoreography(p,c)*
($*$ Determines the impact of removing the participant p from the choreography c $*$)
Input: Participant p, Choreography c
Output: A change impact graph denoting to which extent choreography c and its subchoreographies are impacted by the removal of p

1. $changeImpactGraph \leftarrow$calcChangeImpactGraph($p$,$c$)
2. $changeImpactGraph$.getVertex(p).setImpactChance(1.0)
3. $topSort \leftarrow$calcTopologicalSort($changeImpactGraph$)
4. $topSort$.remove(p)
5. **for** $i \leftarrow 0$ **to** $topSort$.size() -1
6. **do** $(sub)chor \leftarrow topSort[i]$
7. $impactChance \leftarrow$calcImpactChance($(sub)chor$,$changeImpactGraph$)

8. $changeImpactGraph$.getVertex($(sub)chor$).setImpactChance($impactChance$)
9. **return** $changeImpactGraph$

The first step concerns building the *change impact graph* of p in c (line 1). However, in order to calculate such graph, we first need the *subchoreographies tree* of c (Figure 5.9a). In such tree, vertices denote (sub)choreographies and edges denote calling relationships between them. For instance, if a choreography C_i invokes a subchoreography $C_i.Sub_i$, then we create vertices C_i and $C_i.Sub_i$, and draw an edge from C_i towards $C_i.Sub_1$.

Once we have the subchoreographies tree, we are position to calculate the change impact graph of p in c (Figure 5.9(b)). This graph denotes which subchoreographies of c can be impacted by the removal of p. The participant p (which is the only vertex without outgoing edges) and the choreography c are also included in this graph. In particular, the (direct) predecessors of p are precisely those (sub)choreographies that have at least one choreography task whose participants include p.



(a) Subchoreographies Tree of c (b) Change Impact graph of p in c

Figure 5.9: Intra-choreography analysis: Change impact graph of p.

The algorithms for calculating the change impact graph of p in c and the subchoregraphies tree of c are presented below:

Algorithm *calcChangeImpactGraph(p,c)*
Input: Participant p, Choreography c
Output: The impact impact graph of p in cp
1. $h \leftarrow$calculateSubchorTree(c)
2. h.addVertex(p)
3. **for** each (sub)chor c in h
4. **do if** (c uses p)
5. **then** h.addEdge(c,p)
6. h.reverseEdges()
7. $s \leftarrow$reachable vertices from p in h (dfs)
8. $g \leftarrow$subgraph of h induced by the set of vertices s
9. g.reverseEdges()
10. **return** g;

Algorithm *calculateSubchorTree(c)*
Input: Choreography c
Output: Subchoreographies tree t
1. $t \leftarrow$new Tree()
2. t.addVertex(c)
3. $q \leftarrow$new Queue()

4. q.add(c)
5. **while** (q is **not** empty)
6. **do** $c \leftarrow q$.poll()
7. **for** each subchoreography sub in c
8. **do** t.addVertex(sub)
9. t.addEdge(c,sub)
10. q.add(sub)
11. **return** t

We now go back to the main algorithm. Once we have the change impact graph of p in c, we need to decorate it with the actual chances of impact for each vertex. We accomplish this by obtaining the set of possible execution paths from each (sub)choreography and then determining whether p takes part in at least one choreography task inside such paths. The main idea is that if p takes part in all paths of a certain subchoreography sub_1, then the chances of sub_1 being impacted is one. Analogously, if p takes part in none of the execution paths, then the chances of sub_1 being impacted is zero. Since choreographies may include subchoreographies in their execution paths, we calculate the chances of impact according to the topological sort of the change impact graph (lines 3 and 4). This is illustrated in Figure 5.10.

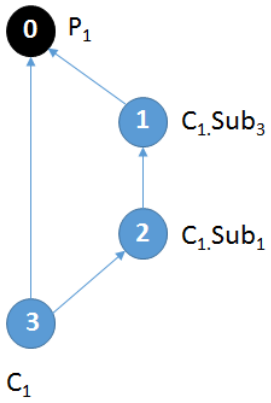


Figure 5.10: Change impact graph of p in c: Topological Sort

Therefore, the chances of each (sub)choreography being impacted by a change of p (line 7 from main) are determined according to the topological sort of the change impact graph of p in c. In the following, we present the algorithms for calculating the impact chance.

Algorithm *calcImpactChance*(c,$changeImpactGraph$)
Input: Choreography c
Output: Chance of c being impacted by the removal of p
1. $execPaths \leftarrow$discoverExecutionPaths(c)
2. $sumPathImpact \leftarrow 0$
3. **for** each $execPath$ in $execPaths$
4. **do** $pathImpact \leftarrow$calcPathImpact($execPath$,$changeImpactGraph$)
5. $sumPathImpact \leftarrow sumPathImpact + pathImpact$
6. $avgPathImpact \leftarrow sumPathImpact$ / $execPaths$.size()
7. $impactChance \leftarrow avgPathImpact$
8. **return** $impactChance$

Algorithm *calcPathImpact*($execPath$,$changeImpactGraph$)
Input: ExecutionPath $execPath$
Output: The path impact

1. $maxStepImpact \leftarrow 0$
2. **for** each step s from $execPath$
3. **do if** (s.getElement() is in $changeImpactGraph$)
4. **then** $stepImpact \leftarrow changeImpactGraph$.getVertice($s$.getElement()).getImpactChance()
5. **if** ($stepImpact > maxStepImpact$)
6. **then** $maxStepImpact \leftarrow stepImpact$
7. $pathImpact \leftarrow maxStepImpact$
8. **return** $pathImpact$

Q4) *If I removed or changed the signature of a certain participant operation O_i, then what would be the impact on the choreography?*

The algorithm is analogous to the previous one. The only difference is that instead of determining which (sub)choreographies rely on a certain participant (when building the change impact graph), we now determine the set of (sub)choreographies that contain a task defined in terms of the participant's operation O_i.

Q5) *If I changed the behavior of a certain participant operation, then what would be the impact on the choreography?*

The algorithm is analogous to the one developed for Q4. However, in this case, we refine the calculation of path impact. In the original case, the calculation of path impact relies on discovering the step with the highest impact. Also, calculating step impact depends on the chances of impact for each (sub)choreography. We now take one more aspect into account: the position of such step inside the execution path. Steps that occur early in the path receive a higher coefficient, while steps that occur late in the path receive a lower coefficient. We took this approach since we believe that the chances of a choreography C_j being impacted by a choreography C_i are greater when C_j calls C_i right in the begging of its execution. For instance, if C_i happens to have a bug and return an incorrect value to C_j, then all subsequent steps of C_j will be susceptible to wrong behavior. In the extreme case, the first step in C_j would be invoking C_i. In such case, our position coefficient equals to 1. This very same idea was employed in our previous study on workflow repository evolution [20].

Q6) *How can each participant be characterized from the perspective of Social Network Analysis (SNA)?*

In previous deliverables [28, 36], we presented a series of Social Network Analysis (SNA) metrics to help characterize collaborating individuals in a group, including: Eigenvector Centrality and Betweenness Centrality. In order to make this deliverable as self-contained as possible, we briefly present these two metrics again:

Eigenvector Centrality. Degree centrality is defined as the number of ties that a node has [46]. Therefore, Degree Centrality is a *local measure*, as only the connections of a node with its neighbor are taken into account to evaluate its importance. The Eigenvector Centrality is a natural extension to the notion of Degree Centrality: each node firstly awards "one centrality point" for every neighbor it has, and then the Eigenvector Centrality algorithm gives each node a score proportional to the sum of the scores of its neighbors. The key idea is that a node's importance is increased when it has connections to other nodes that are themselves important [46, 17]. Eigenvector Centrality can be computed by an iterative degree calculation procedure known as the *accelerated power method* [13]:

Algorithm *Eigenvector Centrality Calculation*
($*$ Eigenvector Centrality computation via Accelerated Power Method $*$)
Input: An adjacency matrix $A_{i,j}$, where $A_{i,j} = 1$ if the i^{th} node is adjacent to the j^{th} node, and $Ai, j = 0$ otherwise
Output: Eigenvector centrality value for all graph nodes
1. Set $C_E(v_i) = 1$ for all i
2. Compute $C^*_E(v_i) = \sum_j A_{i,j} * C_E(v_j)$
3. Set λ equal to the square root of the sum of squares of each $C^*_E(v_i)$
4. Set $C_E(v_i) = C^*_E(v_i)/\lambda$ for all i
5. Repeat lines 2 to 4 until λ stops changing

Betweenness Centrality. Nodes that occur on many shortest paths (a.k.a. geodesic distance) between other vertices have higher betweenness than those that do not. Hence, betweenness centrality evaluates the degree of control a node has over the information flowing in the network. Messages sent through the network frequently pass through these nodes, i.e. they act as "brokers." Betweenness Centrality is given by the following equation

$$C_B(v) = (\sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}})/[(n-1)*(n-2)],$$

where σ_{st} is the number of shortest paths from s to t, and $\sigma_{st}(v)$ is the number of shortest paths from s to t that pass through a vertex v.

In the context of intra-choreography analysis, these two metrics provide insights into how the different participants collaborate to achieve the choreography goal. Our tool addresses the following subquestions:

Q6.a) *How important is a certain participant in a choreography?*

Choreographies include participants that interact with each other. A participant that receives many requests from other participants is naturally important/central to the choreography. As a consequence, changing the behavior of such important participant may directly impact the other participants and the choreography as a whole. We operationalize the idea of importance using the Eigenvector Centrality metric. More precisely, we build the participants call-graph (Algorithm *calcParticipantsCallGraph(c)*) and calculate the Eigenvector Centrality score of each vertex in the network. We perform a statistical quartile analysis of the score distribution and we deem those values higher than or equal to the third quartile (Q3) as high. In other words, vertices (participants) with score higher than or equal to Q3 are considered important.

Q6.b) *Are there participants who act as brokers/bridges/hubs?*

Participants with high Betweenness Centrality somehow control the flow of information in the choreography, acting as brokers/bridges/hubs. Analogously to the previous case, we build the participants call-graph, calculate the centrality score of each vertex in the network, and perform a quartile analysis of the distribution to spot participants with high betweenness.

Q7) *Ok, there are many analyses I can perform to better understand the characteristics of a certain choreography C_i. Is there a way to summarize this information?*

Indeed, choreography designers might find it useful to have a summary report about a certain choreography. Therefore, our tool features a summary report generation. This report includes two key sections: an interactive visualization of the participants call-graph and useful statistics for C_i. Such statistics include: the number of subchoreographies and participants, statistics for each participant (e.g., the number and percentage of provided operations, required operations, clients, and providers), and statistics for each participant's operation (e.g., number of clients, etc.).

Having answered the questions related to a single choreography, we now switch the focus to *inter-choreography analysis*. This time, we take *call choreography* relationships into account and investigate the overall impact of changing a certain choreography. We address the following questions:

Q8) *If I removed a certain choreography C_i, then which other choreographies would be possibly impacted? What would be the impact level on these other choreographies?*

In the ever-changing business environment, organizations continuously refine their processes to benefit from and meet the constraints of new technology, new business rules, and new market requirements [10]. Therefore, it is natural to assume that choreographies will also change over time. Indeed, let's consider the particular context of the Future Internet (FI). In this context, choreographies (e.g., the ones from the "Passenger-friendly airport" UC) will likely be stored in choreography registries (such as ServicePot [37, 45]). To avoid repetition and developing things from scratch, developers will likely evolve their inter-organizational business processes by building new choreographies on top of those from the registry (e.g., including a new choreography to cope with a new scenario of the "Passenger-friendly airport" UC). If choreographies become too entangled, two problems might occur. First, choreography developers may become reluctant to apply changes to choreographies. In this case, the choreographies inside the registry would become *less flexible*, since they would neither leverage opportunities nor deal with the constraints of new technology, new market requirements, and new laws [10]. Second, choreography developers may end up performing changes to choreographies without paying attention to the associated impact, because it is too difficult to be aware of all interdependencies and evaluate how critical they are. In this case, the choreographies inside the registry would become *less reliable*, since ad-hoc changes are likely to induce side and/or ripple effects In fact, previous research already showed that making software changes without visibility into their effects can lead to poor effort estimates, delays in release schedules, degraded software design, unreliable software products, and premature retirement of software systems [26, 16, 11]. In summary, by being less flexible and less reliable, the registry also becomes *less evolvable*.

The following algorithm describes how we determine the impact of removing a certain choreography c from a registry/repository r of choreographies.

Algorithm *calcImpactOfRemovingChoreography(c,r)*
($*$ Determines the impact of removing the choreography c from the repository of choreographies $*$)
Input: Choreography c, Repository r
Output: A change impact graph denoting to which extent repository choreographies are impacted by the removal of c

1. $changeImpactGraph \leftarrow$calcChangeImpactGraph($c$,$r$)
2. $changeImpactGraph$.getVertex(c).setImpactChance(1.0)
3. $topSort \leftarrow$calcTopologicalSort($changeImpactGraph$)
4. $topSort$.remove(c)
5. **for** $i \leftarrow 0$ **to** $topSort$.size() -1
6. **do** $c \leftarrow topSort[i]$
7. $calledChors \leftarrow changeImpactGraph$.getSuccessors($c$)
8. $calledChorsMap \leftarrow$Map $< K, V >$ where K is a called choreography and V is the respective impact chance obtained from $changeImpactGraph$

9. $g \leftarrow$calcImpactOfRemovingCalledChorsFromChoreography($calledChorsMap$,c)
10. $impactChance \leftarrow g$.getVertex($c$).getImpactChance()
11. $changeImpactGraph$.getVertex(c).setImpactChance($impactChance$)
12. **return** $changeImpactGraph$

The first task consists of determining the change impact graph of c in r (line 1 from main). The algorithm *calcChangeImpactGraph(c,r)* describes the way we build such graph. The first task (line 1) involves calculating the *choreographies call-graph* h. The algorithm *calcChoreographiesCallGraph(r)* describes the way we build this other graph. In a few words, we traverse all choreographies in the repository and add a directed edge from c_i to c_j in h if and only if c_i calls c_j via a *call choreography* construct. Once we finish building h, we reverse its edges (line 2) and discover the vertices (choreographies) reachable from c (line 3). This set of vertices s denotes the choreographies that are possibly impacted by a change in c. We then obtain the change impact graph of c in r by calculating the subgraph of h induced by s (line 4) and reversing its edges (line 5). In Figure 5.11, we show an example of a change impact graph extracted from a hypothetical choreographies call-graph.

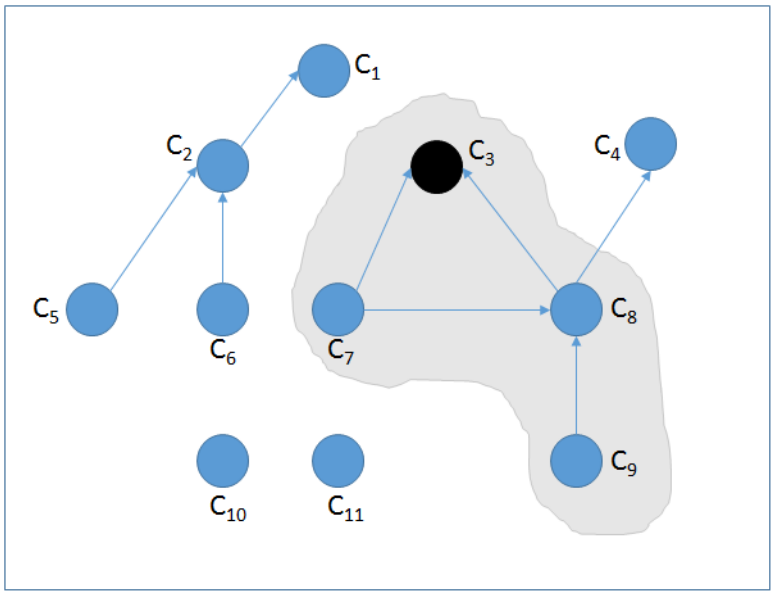


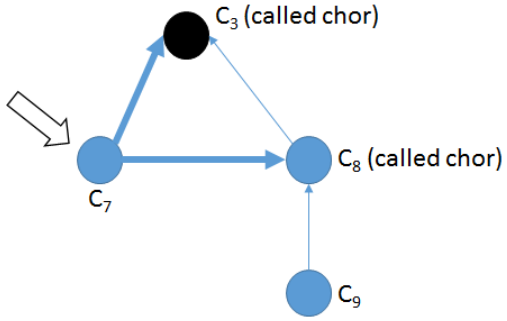
Figure 5.11: Change impact graph of C_3 extracted from a hypothetical choreographies call-graph

Algorithm *calcChangeImpactGraph(c,r)*
Input: Choreography c, Repository r
Output: The graph g denoting the repository choreographies which can be impacted by a change in c
1. $h \leftarrow$calcChoreographiesCallGraph(r)
2. h.reverseEdges()
3. $s \leftarrow$reachable vertices from c in h (dfs)
4. $g \leftarrow$subgraph of h induced by the set of vertices s
5. g.reverseEdges()
6. **return** g;

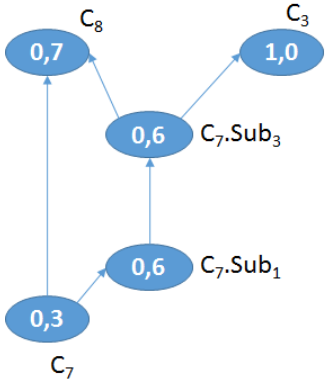
Algorithm *calcChoreographiesCallGraph(r)*
Input: Repository r with all choreographies
Output: Choreographies call graph g
1. $g \leftarrow$new DirectedGraph()
2. **for** each choreography c in the repository r
3. **do** $calledChors \leftarrow c$.getCalledChoreographies()

```
4.          if (calledChors is not empty)
5.            then if (g does not contain vertex c)
6.                   then g.addVertex(c)
7.                 for each calledChor in calledChors
8.                   do if (g does not contain vertex calledChor)
9.                        then g.addVertex(calledChor)
10.                     g.addEdge(c,calledChor)
11. return g
```

Once we have the change impact graph of c in r, we can return to the main algorithm. We set the impact chance of c to one (line 2) and determine the impact chances for the remaining vertices in a topological order (lines 2-6). For each choreography c_i, we determine its direct successors (i.e., the choreographies it calls) and build a map where the *key* corresponds to a successor of c_i and the *value* corresponds to the successor's impact chance obtained from the change impact graph (lines 7 and 8). In the next step, we calculate the impact of removing the called choreographies from c_i (line 9). This step invokes a routine analogous to the algorithm *calcImpactOfRemovingParticipantFromChoreography(p,c)*. However, instead of dealing with the removal of a single participant, the routine deals with a set of choreographies (i.e., the called choreographies). Moreover, instead of attributing an impact chance of one to the participant, it now attributes each choreography with its corresponding value from map. This routines return a graph decorated with the chances of impact for c_i (and its subchoreographies), as illustrated in Figure 5.12.



(a) C_7 in the change impact graph of C_3



(b) The chances of C_7 (and its subchoreographies) being impacted: illustrative values

Figure 5.12: Intra-choreography analysis: Change impact graph of p.

The main algorithms finishes after determining the impact chance for every choreography in the change impact graph.

Q9) *If I changed the structure of a certain choreography C_i, then which other choreographies would be possibly impacted? What would be the impact level on these other choreographies?*

The algorithm is analogous to the previous one that addresses Q8. However, as in the case of Q5, we refine the calculation of path impact using a coefficient.

In fact, based on all algorithms we developed, it becomes relatively ease to answer the following three other questions:

- **Q10)** *If I removed a certain participant P_i, then what would be the impact on choreographies?*
- **Q11)** *If I removed or changed the signature of a certain participant operation O_i, then what would be the impact on choreographies?*
- **Q12)** *If I changed the behavior of a certain participant operation, then what would be the impact on choreographies?*

Q13) *What is the importance of each choreography in the repository?*

Analogously to what we did for Q6.a, we calculate the choreographies call-graph and determine the Eigenvector Centrality score for each vertex.

Q14) *What is the stability of the choreography repository?*

We tackle this question by building the choreographies call-graph and calculating its stability. We recall the definition of stability provided in D2.1 [28]:

> **Overall stability.** Overall stability refers to the average propagation level of changes performed in the nodes of a network. The higher the stability, the less changes tend to propagate. In object-oriented systems, this measure is employed to determine whether the overall coupling between elements is being controlled. We calculate stability as follows:
>
> $$Stability(G) = 1 - AvgImpact(G),$$
>
> $$AvgImpact(G) = [\sum_{v \in V} \gamma(v)]/n^2$$
>
> , where $\gamma(v)$ denotes the size of the transitive closure of v (graph vertices from which v can be reached).
>
> For instance, a stability level of 70% means that a change affects, in average, 30% of all existing network nodes.

Q15) *Ok, there are many analyses I can perform to better understand the characteristics of choreographies. Is there a way to summarize this information?*

Indeed, choreography designers might find it useful to have a summary report about all choreographies included in a certain registry/repository. The report includes a treemap visualization of the change impact of each choreography. Treemap is an efficient and compact visualization method that uses nested rectangles to display information with hierarchical characteristics [25]. An example is depicted in Figure 5.13.

We leverage the hierarchical nature of treemaps to display the subchoreographies tree of each choreography. We calculate Q9 for each choreography and color them according to the results: red

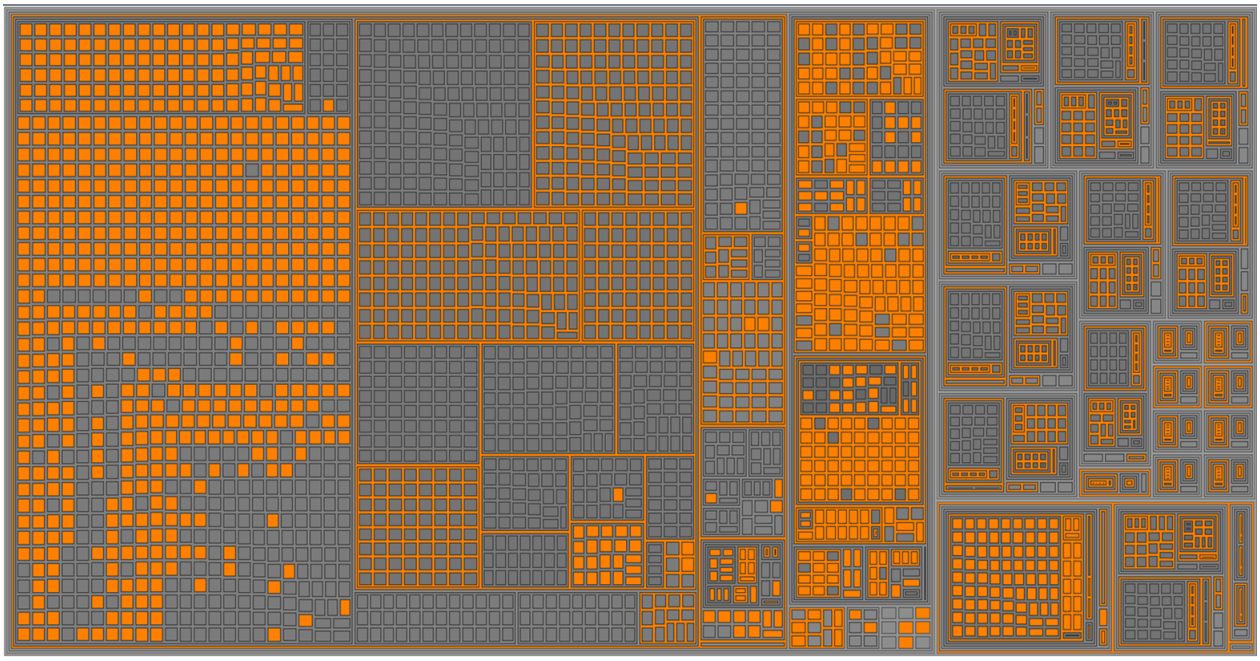


Figure 5.13: An example of a treemap: non-leaf rectangles denote folders and leaf rectangles denote files. Orange rectangles denote the folders and files changed by a certain developer over a certain period of time.

(sub)choreographies have high change impact, yellow (sub)choreographies have medium change impact, and green (sub)choreographies have low change impact. An example is shown in Figure 5.14. This enables choreography developers to quickly spot choreographies with high change impact.

Besides that, we also provide useful statistics, such as the importance of each choreography (Q13) and the stability of the choreography repository (Q14). Finally, choreography developers can also visualize the change impact graph of a chosen choreography.

After Service Binding

We now switch the focus to the analyses we can perform after services are bound to the choreography, i.e., after the `ChoreoSpec` file is generated by the synthesis processor. Among other things, this file includes two key pieces of information: (i) the roles played by a service (ii) which other services were picked to fulfill a certain service dependencies. Since the synthesis process is fully automated, the choreography designer might want to be aware of how the choreography ended up being realized. In particular, the designer might be interested to know which services were selected to realize the choreography and how such depend on one another. In the following, we present the change impact analysis questions covered by our tool.

Q16) *If I removed a choreography bound service, which other services from the same choreography would be impacted?*

The dependency analyzer tackles this problem by providing a visualization of service dependencies. In particular, the tool offers the following features:

- **Visualizing service dependencies for increased awareness of how the choreography is being realized.** The tool parses the `ChoreoSpec` file, extracts dependencies, and builds a service dependency graph. This graph is interactive, so choreography developers can move nodes around and inspect dependencies. Our tool also provides useful information about the service dependency graph, including the number of services, the number of links, and its stability.

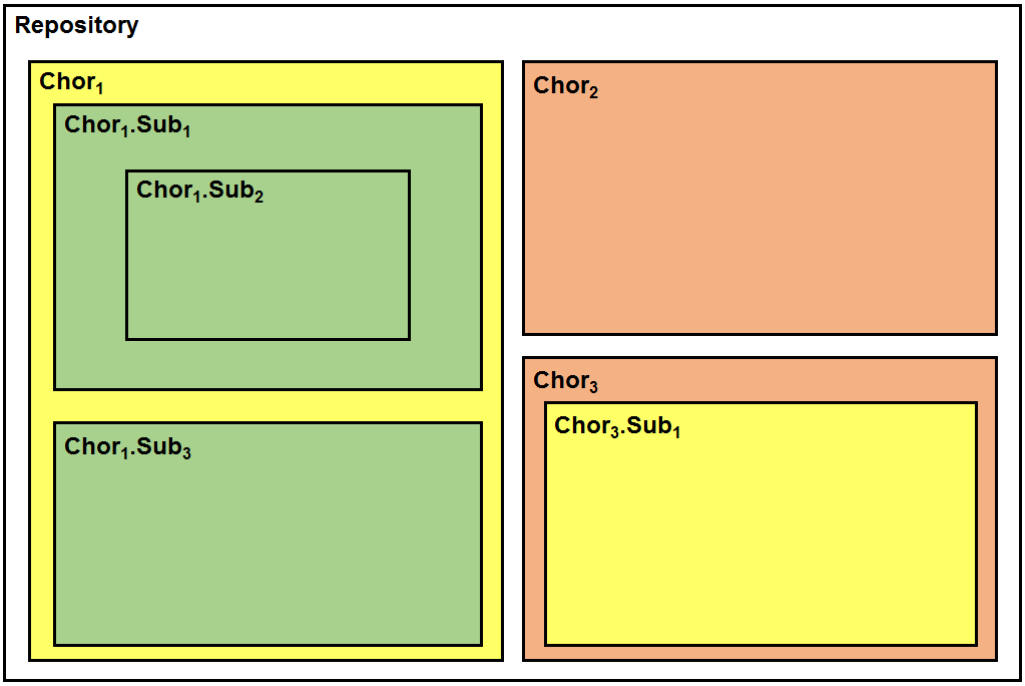


Figure 5.14: The treemap of a hypothetical choreography repository/registry

- **Determining the importance of each service bound to the choreography.** If a choreography developer selects a certain service, the tool shows dependency-related information about it, including the number of incoming dependencies, the number of outgoing dependencies, the relative importance of the service (given by the Eigenvector Centrality measure), and the number of services it may possibly affect when changed (ripple effect number).
- **Visualizing ripple effect.** The dependency analyzer also enables choreography developers to visualize ripple effect. When the end user chooses to visualize the ripple effect of a particular service, the tool highlights all possibly affected services in the interactive graph.

Q17) *If I needed to take down a certain choreography bound service, which other enacted choreographies would be impacted?*

This question involves assessing the overall impact of changing a choreography bound service. The following algorithm answers this question in terms of *choreographies that will break* and *choreographies that will need service substitution*:

1) Calculate choreographies call-graph.
2) Use the Enactment Engine API to build Table 5.6. Inspect such table and discover which choreography participants would be impacted by a change on the chosen service.

Participant	Implementing Service
$Chor_1.P_3$	S_1
$Chor_2.P_4$	S_1
$Chor_1.P_1$	S_2

Table 5.6: Service x Participant table (T1)

3) For each of the impacted participants, use the synthesis processor and the AoSBM to check whether there is an alternate service that could play its role in the choreography C.

4) If there is no suitable substituting service, then the choreography C will break. If there at least one suitable substituting service, then choreography C will require service substitution/adaptation.

5.2.3. Dependency Analyzer in Action

The Dependency Analyzer is part of the Choreography Analyzer subsystem, which provides functionalities for conducting change impact analysis and predicting QoS. The dependency analyzer is a J2SE standalone tool. It is easily compilable using Maven 3 and requires Java 6.0 or later. Installation instructions and details at the tool official website: `http://choreos.eu/bin/view/Documentation/depanalyzer`.

Since we already analyzed BPMN2 diagrams using the QoS Analyzer, we will now focus on analyzing the impact of changing choreography bound services. To this end, we will analyze the `ChoreoSpec` generated by the synthesis processor for the "In-store Marketing and Sale" scenario from the "Adaptive Customer Relationship Booster (ACRB)" WP7 use case (see the Listing A.1 in Appendix A.1):

The UML diagram in Figure 5.15 depicts the dependencies among services embedded in this `ChoreoSpec`[3]. Names in parenthesis indicate the role played the service.

We now investigate the effects of removing a choreography bound service on other services from the same choreography (Q16).

Visualizing service dependencies for increased awareness of how the choreography is being realized. . Right after the `ChoreoSpec` file is loaded into the tool, it shows the service dependency graph (Figure 5.16). On the left-hand side, it shows useful information about the choreography: number of services, number of links, and stability. In particular, the stability of this choreography is 52,6%, which means that the overall coupling is a little high.

At this point, the choreography designer can investigate dependencies by moving nodes and edges around. The choreography designer can also use the interactive commands shown in the bottom to manipulate the graph (zoom in, zoom out, refresh layout, etc).

Determining the importance of each service bound to the choreography. When the choreography designer selects a certain service, the tool shows useful information about it on the left-hand side panel. In Figure 5.17, we can see that the `marketingapplication` relative importance is 14.25% (the highest among all services) and that it might impact 66.6% of all services when it is changed. This information reveal that such service is core to the choreography and should be carefully tested and maintained/evolved.

Visualizing ripple effect. We conclude our analysis with a visualization of the ripple effect that might occur as the result of a change applied to `marketingapplication`. As depicted in Figure 5.18, the tool highlights all possibly affected services.

As a concluding remark, we believe our approach advances the body of knowledge on choreography analysis and complements runtime analyses (e.g., QoS monitoring). Our approach should also help organizations build more flexible and reliable choreographies in the environment of the Future Internet.

[3]For more readability, in the figure the dependencies among CDs are not shown. These dependencies serves to permit CDs to exchange additional communication, i.e., coordination information, as described in Chapter 4

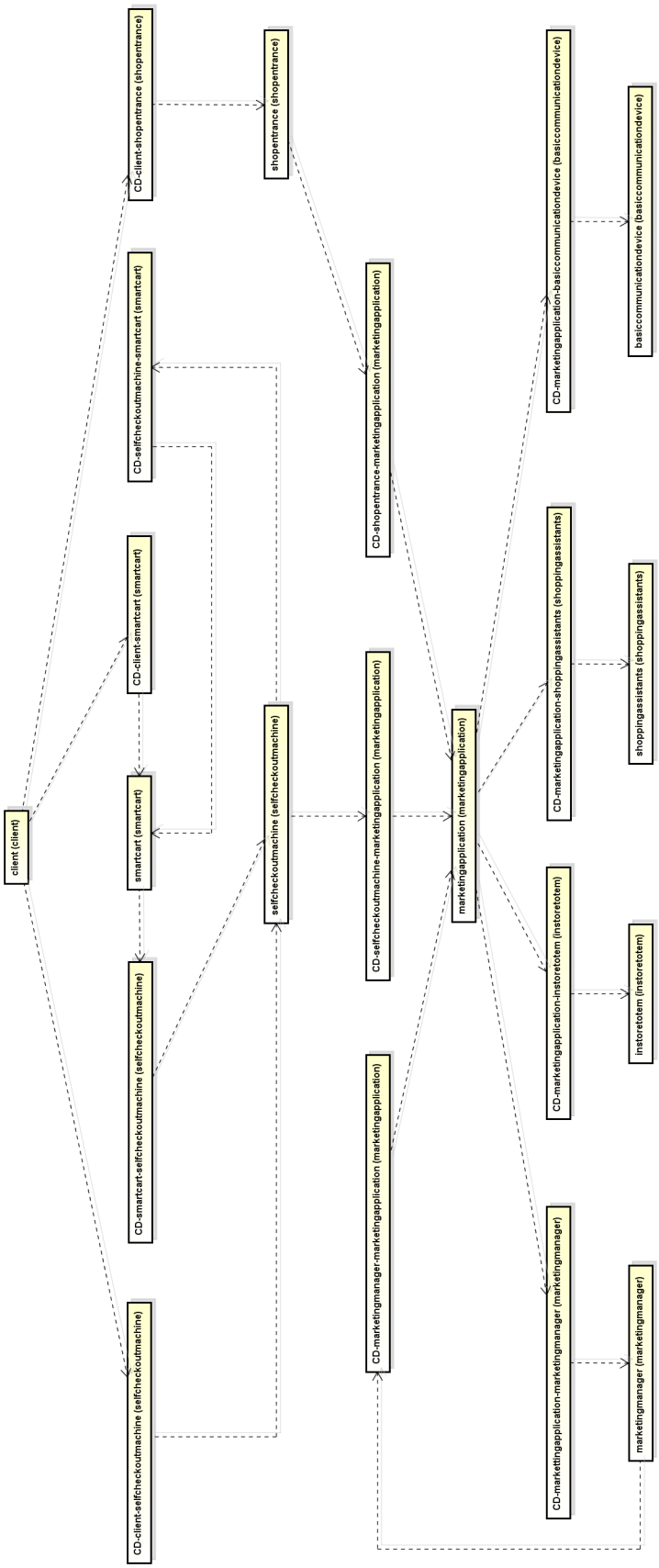


Figure 5.15: UML diagram depicting the service dependencies embedded in the `ChoreoSpec` file

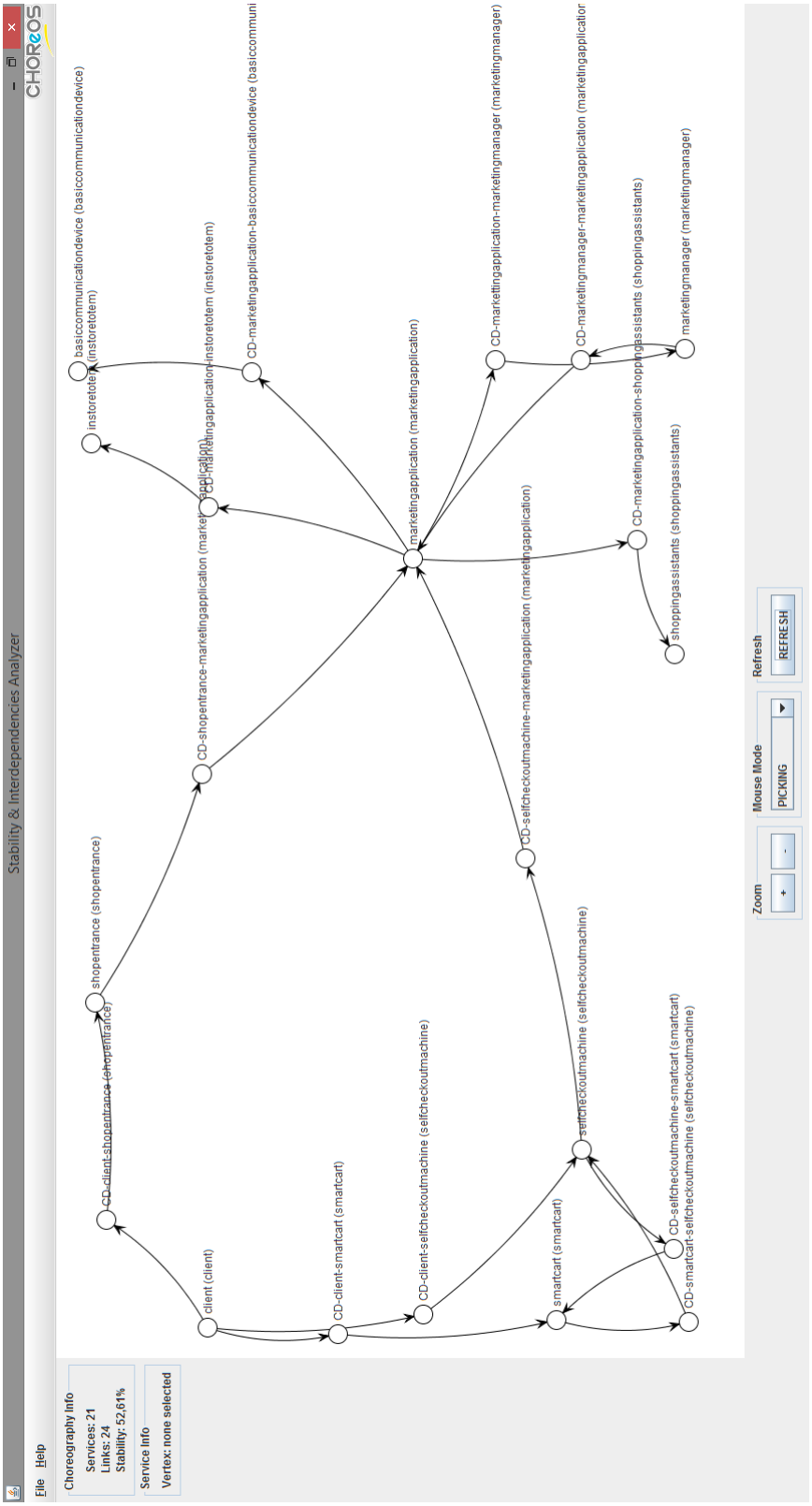


Figure 5.16: Our tool running the service dependency analysis

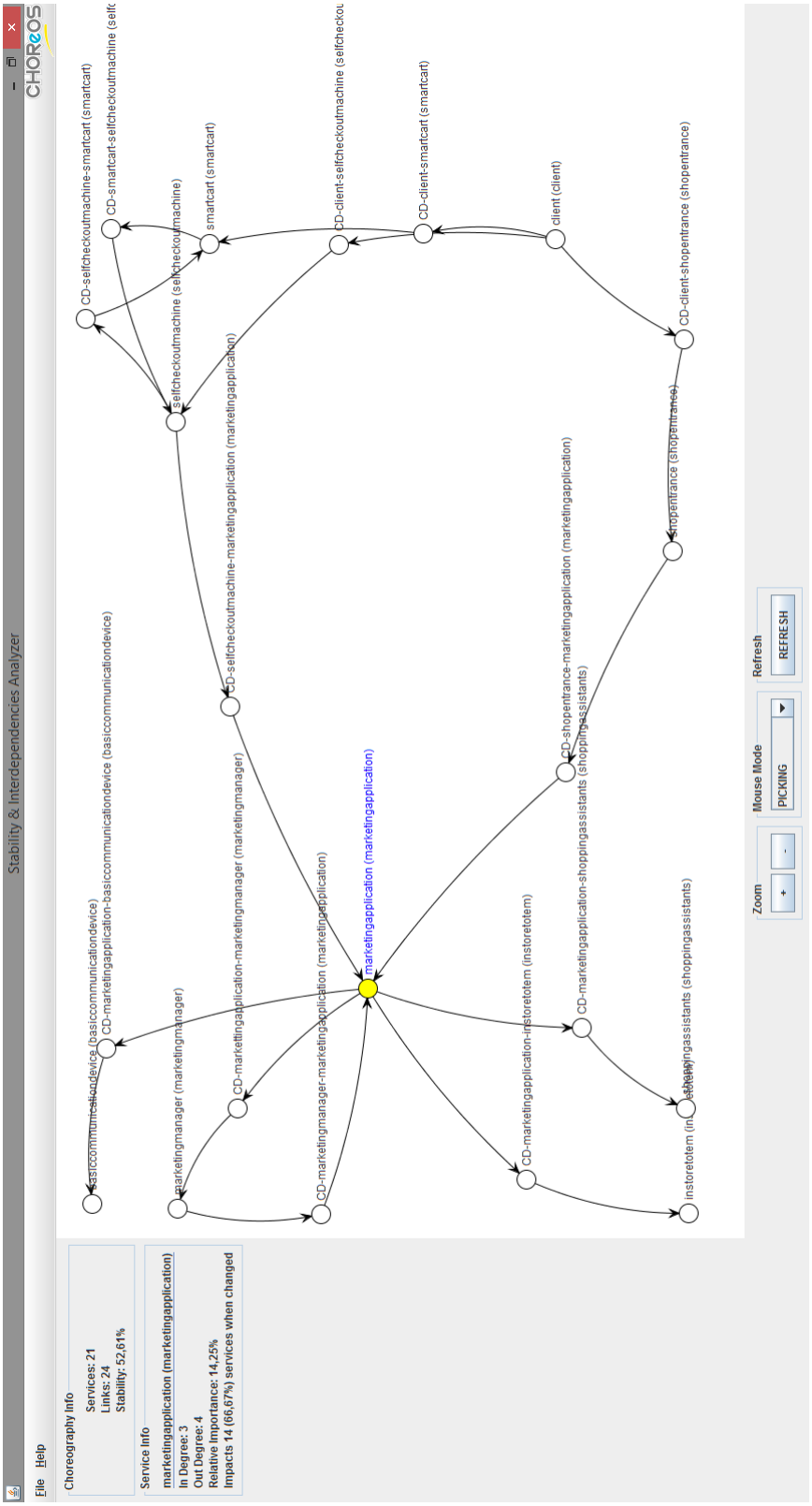


Figure 5.17: Our tool running the service dependency analysis

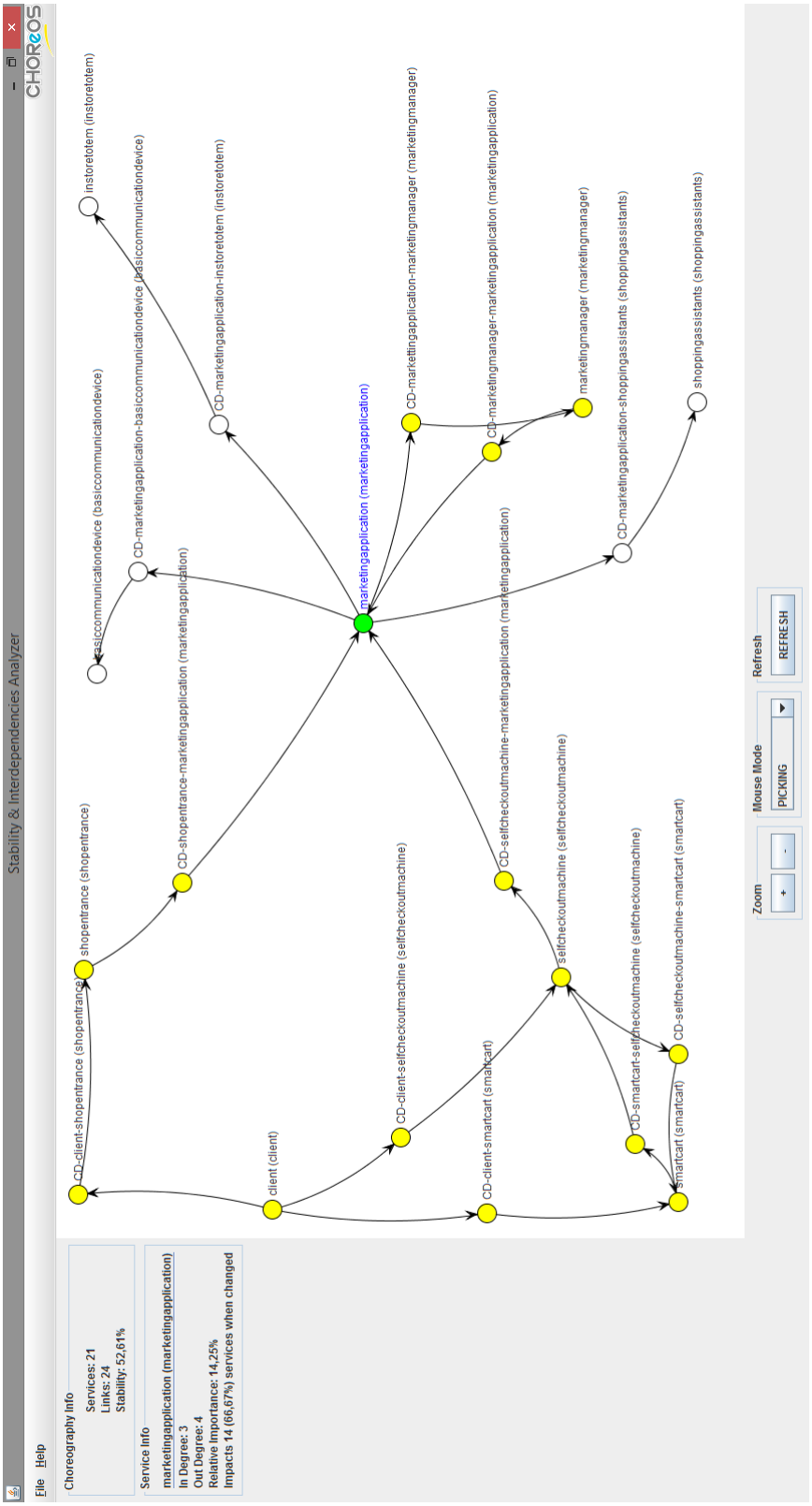


Figure 5.18: Our tool running the service dependency analysis

6 Conclusions and Future Work

In this deliverable we have described the software that have been developed in WP2 in order to implement the final CHOReOS development process defined in D2.2 [36]. The released tools are made available from the official CHOReOS repository hosted by OW2 and are integrated in the CHOReOS IDRE. In particular, we have described (i) the domain expert requirements specification tool, (ii) the large scale service base, its management, its registration and query engine, (iii) the synthesis processor REST services and the Eclipse plugins, and (iv) the analysis tools.

Domain expert requirements specification tool – The latest version of the CHOReOS Requirements Tool builds upon the Excel-based prototype that was first presented in D2.1 [28]. It was designed and developed to explicitly support the domain expert specification framework and the early stages of the design choreography specification process presented in D2.2 [36]. In particular, the tool is now fully integrated with MagicDraw, the BPMN2 modelling environment, with outputs from the requirements process being provided to inform choreography design and the specification of non-functional properties in Q4BPMN.

We have applied the Requirements Tool to two of the use cases, the passenger-friendly airport and DynaRoute. In the latter, in particular, we have successfully demonstrated the end-to-end choreography specification process using the integrated tools and third party services defined in Section 2.1. The Requirements tool has been shown to be usable and scalable, however, improvements to usablity have been identified and reported in D10.3 [43], and ideas for improving the tool's scalability have been reported in Section 2.3 of this deliverable. Furthermore, whilst the functions for creating requirements clusters were effective, they were not efficient. Suggested improvements to the use of the similarity algorithm service are reported in Section 2.3. Finally, the successful integration of the Requirements Tool with MagicDraw was a significant achievement for the choreography specification process. An expert usability review of the integration tasks undertaken by the user is presented in D10.3 [43].

Aside from usability improvements to the tool, future work needs to further investigate the mapping between quality requirements and the expression of non-functional systems engineering properties on choreography diagrams. We have made some sound progress in this area, however, we need to be able to better support the domain expert and choreography designer in dealing with different levels of requirements granularity. Satisfaction arguments, as presented in D2.2 [36], provide a means to reason about the relationships between user expressed quality requirements and lower-level system requirements. We need to explore effective ways of implementing this or a similar approach in the modelling environment, currently MagicDraw. Beyond the specification of choreographies, the Requirements Tool could be modified and extended to support other requirements-led approaches.

Large scale service base: management, registration and query engine – The abstraction-oriented service base management facilities enable service discovery in the context of the FI, based on the concept of service abstractions, which represent groups of services that provide similar functional/non-functional properties. In D2.1 [28] we proposed the clustering algorithms that extract abstractions from available service descriptions and exploited the extracted abstractions as a means for organizing similar service descriptions. In D2.2 [36] we proposed WSBQL, a query language that is tailored to the concept of abstractions and a corresponding query engine that enables service lookup queries. In this deliverable we detailed the extensions of our work, and specifically the REST API that allows using the proposed approach in a distributed setting. We further assessed the proposed approach from a

qualitative perspective in the context of the WP6 and WP7 use cases.

Concerning the perspectives of the proposed approach, an issue for future research is the investigation of different clustering algorithms for the extraction of service abstractions. Another interesting issue is to make the proposed approach more interactive, by allowing the end-users to manipulate information that relates to the extracted service abstractions (e.g., the abstract interface specifications, the mappings between the abstract interface and the interfaces of the represented services). Finally, the proposed approach can be extended with personalization features that account for the end-users preferences, positive, or negative feedback, etc.

Choreography synthesis processor – The latest version of the CHOReOS Synthesis Processor, and its underlying methodology, presented in Chapter 4 thoroughly refines and extends our previous results reported in Deliverable D1.3 [33] and Deliverable D1.4(b) [41]. In particular, the processor is now able to handle more complex constructs of BPMN2 Choreography Diagrams, e.g., conditional exclusive gateways (decision, alternative paths), inclusive gateways (inclusive decision, alternative but also parallel paths), parallel gateways (creation and merging of parallel flows), standard and sequential loops.

A set of REST services have been implemented to realize the Synthesis Processor. Moreover, although the processor is a fully automatic batch process, as part of the overall process supported by the CHOReOS IDRE, we have also implemented a set of Eclipse plugins that, connecting to the REST services, allow developers to transform BPMN2 choreography specifications, to edit, visualize (by using the dedicated GMF-based graphical editor aptly developed), save, project, simulate Choreography Labelled Transition Systems (CLTSs), etc. The synthesis plugins allow the user to choose if to execute whether the Automatic Synthesis Processor that (calling in sequence all the Synthesis REST services) produces all the artefacts in one step, or the Interactive Synthesis Processor that produces the artefacts step by step. The plugins can easily be installed on the ECLIPSE platform by using the dedicated update site: `http://choreos.disim.univaq.it/updatesite/site.xml`.

We have applied the synthesis processor to all the CHOReOS use cases. We have proven correctness of the defined algorithm with respect to preventing those interactions that do not belong to the choreography specification. To this end, we have rigorously characterized the notion of undesired (message-oriented) interaction. The application to these use cases has shown that the method is viable and the overhead due to the exchange of coordination information (i.e., Additional Communication) is negligible while coordinating message exchanges. However, beyond the modeling of message-oriented interaction, BPMN2 Choreography Diagrams provide constructs for specifying events. An interesting future work would be to extend our definitions of CLTS, and related Coordinations Models, to account for event-based coordination. As a consequence, the distributed coordination algorithm formalized in Section 4.2.2 must also be revised.

The coordination logic performed by the CDs is *service-independent* since it is based on the expected behavior of the participants as specified by the choreography, rather than on the actual one as obtained after discovery. Within CHOReOS, this is done to consistently realize *separation of concerns*. That is, to separate pure coordination issues (i.e., undesired interactions) from adaptation ones (e.g., syntactic mismatches at the service interface level). The latter can arise whenever a service discovered as a participant does not exactly match the role to be played. In Section 4.2.4, we show how they can be addressed by leveraging our preliminary results reported in [5, 14]. In the future further investigation in this direction is also interesting, towards defining an extended version of CDs that are able to address both coordination and adaptation issues in a *modular* manner, hence still promoting separation of concerns.

Analysis tools – With relation to the *QoS Analyzer*, we currently have solutions that optimize the selection of services based on service response time and energy consumption for Web Service compositions. We proposed solutions for two cases in the optimization: first, the SLAs free case where there is no constraint on the maximal service response time and energy consumption; and second the general case where such constraints are included. Our solutions were based in a MILP approach, which

had affordable processing time for the tested instances. We have some perspectives for continuing this work. The first is to extend the MILP generation on other QoS parameters like the availability and the reputation. The main challenge will consist of translating the aggregation rules into adequate equations. Our second perspective is to develop a global solution including negotiation for the SLAs. As our experiments showed, indeed there are many cases where we have infeasible problems.

With relation to the *Dependency Analyzer*, we have implemented change impact analysis functionalities that rely on techniques described on deliverables D2.1 [28] and D2.2 [36], which include: graph algorithms, social network analysis (SNA) metrics, and visualization techniques. In this deliverable, we present the main use cases of the tool and the associated algorithms we've conceived to support them. This tool is targeted to those that own a series of choreographies and wish to have some guidance and vision into the effects of applying any kind of change to the definition of their choreographies and the services on which such choreographies rely on. As future work, we envision calibrating change impact metrics with the support of runtime data (e.g. with information regarding which paths inside a choreography are executed more often). Besides the preliminary assessments we have already conducted [20], further evaluations using qualitative research would help us validate metrics results and the overall effectiveness of the tool.

The following table summarizes advantages, disadvantages, and possible future enhancements of the WP2 tools:

Tool Name	Pros	Cons	Enhancements
Requirements specification	• Fully integrated with MagicDraw and Q4BPMN • Useable and scalable • Effective creation of requirements clusters	• Not efficient creation of requirements clusters	• Usage of the similarity algorithm service reported in Section 2.3 for improving the efficiency of the requirements clusters creation • Needs to further investigate the mapping between quality requirements and the expression of non-functional systems engineering properties on choreography diagrams
Large scale service base	• Enable service discovery in the context of the FI, based on the concept of service abstractions • Provisioning in a distributed setting via a REST API	• Not interactive • No support for customization	• Investigation of different clustering algorithms for the extraction of service abstractions • Extension with personalization features that account for the end-users preferences, positive, or negative feedback, etc.

Synthesis Processor	• Ability to handle most of the complex constructs of BPMN2 Choreography Diagrams • Formal proof of correctness of the distributed coordination algorithm • Scalable method	• No support for event-based BPMN2 constructs	• Definition of an extended version of CDs that are able to address both coordination and adaptation issues in a modular manner • Extension for handling event-based interactions
Analysis tools (QoS Analyzer)	• Service selection optimized with respect to response time and energy consumption	• Further QoS parameters should be considered	• Extension for taking into account availability and reputation • Developing a global solution including negotiation of SLAs
Analysis tools (Dependency Analyzer)	• Simple and yet powerful change impact analysis framework for service choreographies • Applicable both before and after service binding (generation of `ChorSpec`) • Implemented as an interactive tool that relies on visualization techniques to cope with high amounts of information	• No usability assessment • Does not leverage runtime data to calibrate change impact metrics • Does not provide a REST API	• New change impact analyses were incorporated to the approach • User interface needs further improvements

Table 6.1: WP2 tools summary of achievements

Bibliography

[1] I.F. Alexander and R. Stevens. *Writing Better Requirements*. Addison-Wesley, 2002.

[2] Emil Andriescu, Amel Bennaceur, Paola Inverardi, Valerie Issarny, Romina Spalazzese, and Roberto Speicys-Cardoso. Dynamic Connector Synthesis: Principles, Methods, Tools and Assessment. Research report, December 2012.

[3] Robert S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[4] Marco Autili, Davide Di Ruscio, Amleto Di Salle, Paola Inverardi, and Massimo Tivoli. A model-based synthesis process for choreography realizability enforcement. In *Proceedings of the 16th international conference on Fundamental Approaches to Software Engineering*, FASE'13, pages 37–52, Berlin, Heidelberg, 2013. Springer-Verlag.

[5] Marco Autili, Amleto Salle, and Massimo Tivoli. Synthesis of resilient choreographies. In *Software Engineering for Resilient Systems*, LNCS 8166, pages 94–108. Springer Berlin Heidelberg, 2013.

[6] Cesare Bartolini, Antonia Bertolino, Andrea Ciancone, Guglielmo De Angelis, and Raffaela Mirandola. Quality requirements for service choreographies. In Karl-Heinz Krempels and José Cordeiro, editors, *WEBIST*, pages 143–148. SciTePress, 2012.

[7] Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 795–804, 2011.

[8] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL, pages 191–202. ACM, 2012.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, and Fabio Patrizi. Automatic service composition and synthesis: the roman model. *IEEE Data Eng. Bull.*, 31(3):18–22, 2008.

[10] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data Knowl. Eng.*, 24(3):211–238, January 1998.

[11] Cleidson R. B. de Souza and David F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *Proc. of the 30th International Conference on Software Engineering*, ICSE '08, pages 241–250, New York, NY, USA, 2008. ACM.

[12] 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.

[13] Harold Hotelling. Simplified calculation of principal components. *Psychometrika*, 1:27–35, 1936. 10.1007/BF02287921.

[14] Paola Inverardi and Massimo Tivoli. Automatic synthesis of modular connectors via composition of protocol mediation patterns. In *Proc. of ICSE'13*, pages 3–12, 2013.

[15] NeilA.M. Maiden, SaraV. Jones, Sharon Manning, John Greenwood, and L. Renou. Model-driven requirements engineering: Synchronising models in an air traffic management case study. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering*, volume 3084 of *Lecture Notes in Computer Science*, pages 368–383. Springer Berlin Heidelberg, 2004.

[16] Tom Mens and Serge Demeyer. *Software Evolution*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[17] Mark Newman. *Networks: An Introduction*. Oxford University Press, USA, first edition, 2010.

[18] Yanik Ngoko, Alfredo Goldman, and Dejan Milojicic. An analytical approach for predicting qos of web services choreographies. *Journal of Internet Services and Applications (JISA) - Special issue with best papers from Middleware 2012 workshops*, 2013. Under review.

[19] Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Björn Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Software*, 22(1):32–39, 2005.

[20] Gustavo A. Oliva, Marco A. Gerosa, Dejan Milojicic, and Virginia Smith. A change impact analysis approach for workflow repository management. In *Proceedings of the 2013 IEEE 20th International Conference on Web Services*, ICWS '13, pages 308–315, Washington, DC, USA, 2013. IEEE Computer Society.

[21] D. Papadimitriou. Future Internet–The Cross-ETP Vision Document. *European Technology Platform, Alcatel Lucent*, 8, 2009. http://www.future-internet.eu/fileadmin/documents/reports/Cross-ETPs_FI_Vision_Document_v1_0.pdf.

[22] Pascal Poizat and Gwen Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proceedings of SAC 2012*, pages 1927–1934, 2012.

[23] Gwen Salaün. Generation of service wrapper protocols from choreography specifications. In *Proceedings of the 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 313–322, 2008.

[24] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[25] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, January 1992.

[26] E. Burton Swanson and Cynthia Mathis Beath. *Maintaining information systems in organizations*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[27] CHOReOS Project Team. Integrated choreos middleware and deployment of uls, qos-aware adaptive choreographies - public project deliverable d3.3.

[28] CHOReOS Project Team. CHOReOS dynamic development model definition - Public Project deliverable D2.1, September 2011.

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

[30] CHOReOS Project Team. CHOReOS Mobile-enabled coordination of people Service & Choreographies Design - Public Project deliverable D7.2, September 2011.

[31] CHOReOS Project Team. CHOReOS Passenger Friendly Airport Services Choreographies Design - Public Project deliverable D6.2, September 2011.

[32] CHOReOS Project Team. CHOReOS Perspective on the Future Internet and Initial Conceptual Model - Public Project deliverable D1.2, March 2011.

[33] CHOReOS Project Team. Initial Architectural Style for CHOReOS Choreographies - Public Project deliverable D1.3, September 2011.

[34] CHOReOS Project Team. Requirements and scenarios for the ”Passenger-friendly airport” - Public Project deliverable D6.1, October 2011.

[35] CHOReOS Project Team. CHOReOS Middleware Implementation - Public Project deliverable D3.2.2, September 2012.

[36] CHOReOS Project Team. Definition of the Dynamic Development Process for Adaptable QoS-aware ULS Choreographies - Public Project deliverable D2.2, September 2012.

[37] CHOReOS Project Team. V&V Tools and Infrastructure - Strategies, Architecture and Implementation - Public Project deliverable D4.2.1, April 2012.

[38] CHOReOS Project Team. Adaptive Customer Relationship Booster use case assessment and demonstration, October 2013.

[39] CHOReOS Project Team. Assessment of the DynaRoute pilot deployment and demonstration, October 2013.

[40] CHOReOS Project Team. CHOReOS Courseware - Public Project deliverable D10.3, September 2013.

[41] 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.

[42] CHOReOS Project Team. Final release of the V and V tools and infrastructure, October 2013.

[43] CHOReOS Project Team. Final technical assessment report - Public Project deliverable D10.3, September 2013.

[44] CHOReOS Project Team. Testing and monitoring tools and infrastructure final evaluation report, October 2013.

[45] CHOReOS Project Team. V&V Tools and Infrastructure - Strategies, Architecture and Implementation - Public Project deliverable D4.2.2, January 2013.

[46] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*. Cambridge university press, first edition, 1994.

A Appendix

A.1. Application of the “Synthesis Processor to the In-store Marketing and Sale” scenario (WP7)

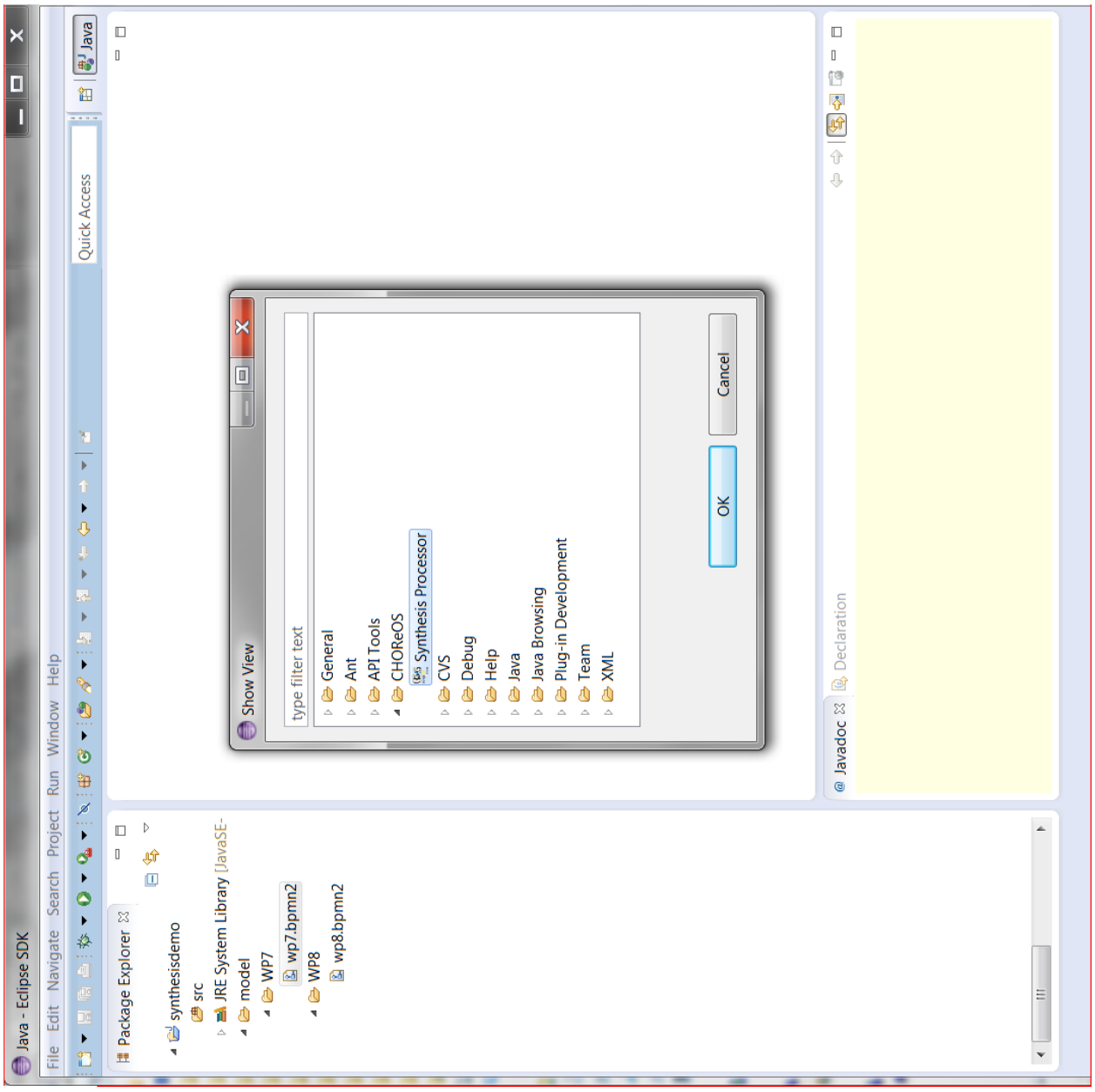


Figure A.1: Opening of the ECLIPSE View dedicated to the CHOReOS Synthesis Process

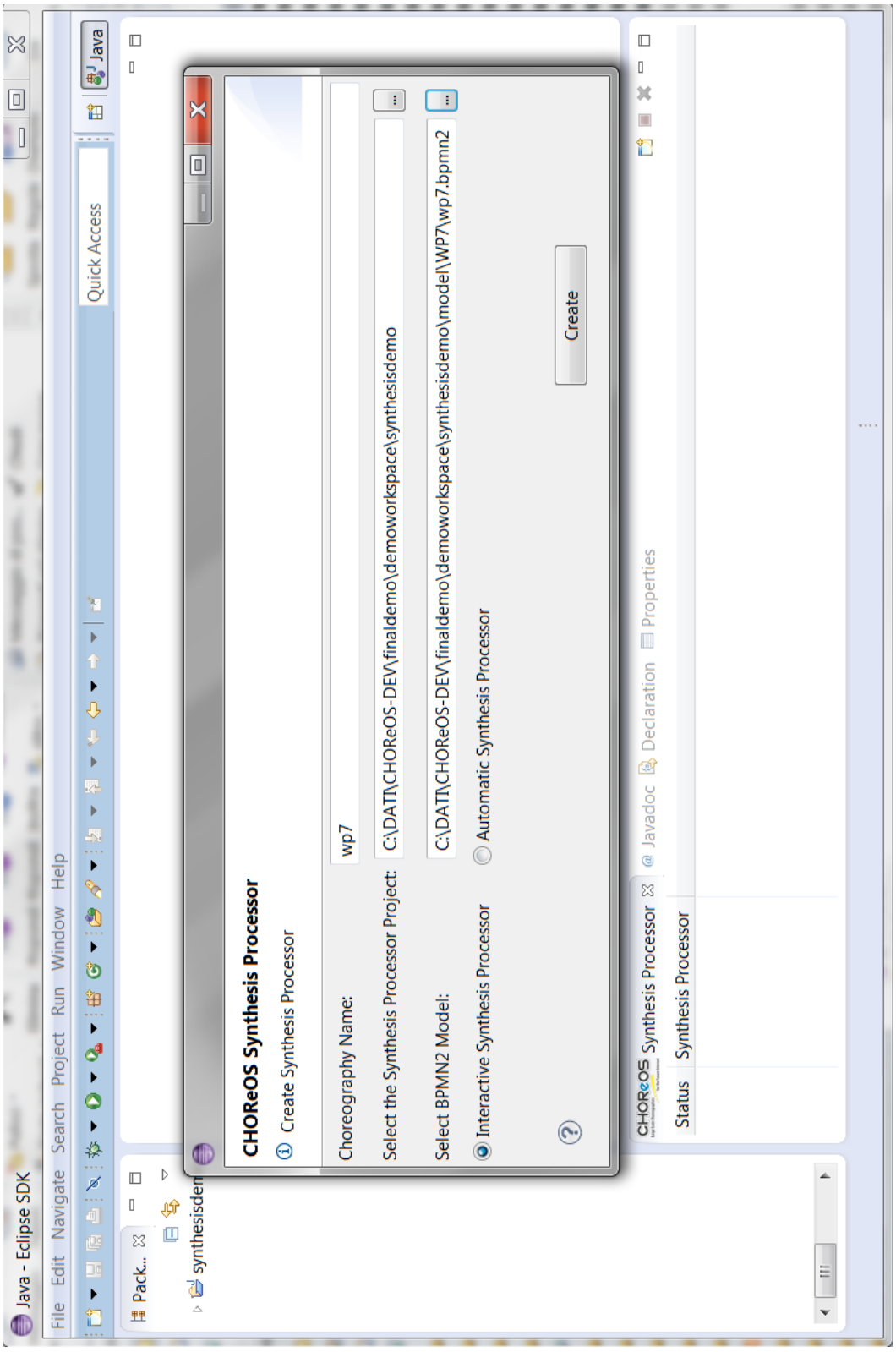


Figure A.2: Creation of a Synthesis Processor project. Note that, the user can choose if to execute whether the Automatic Synthesis Processor that (calling in sequence all the Synthesis REST services) produces all the artefacts in one step, or the Interactive Synthesis Processor that produces the artefacts step by step.

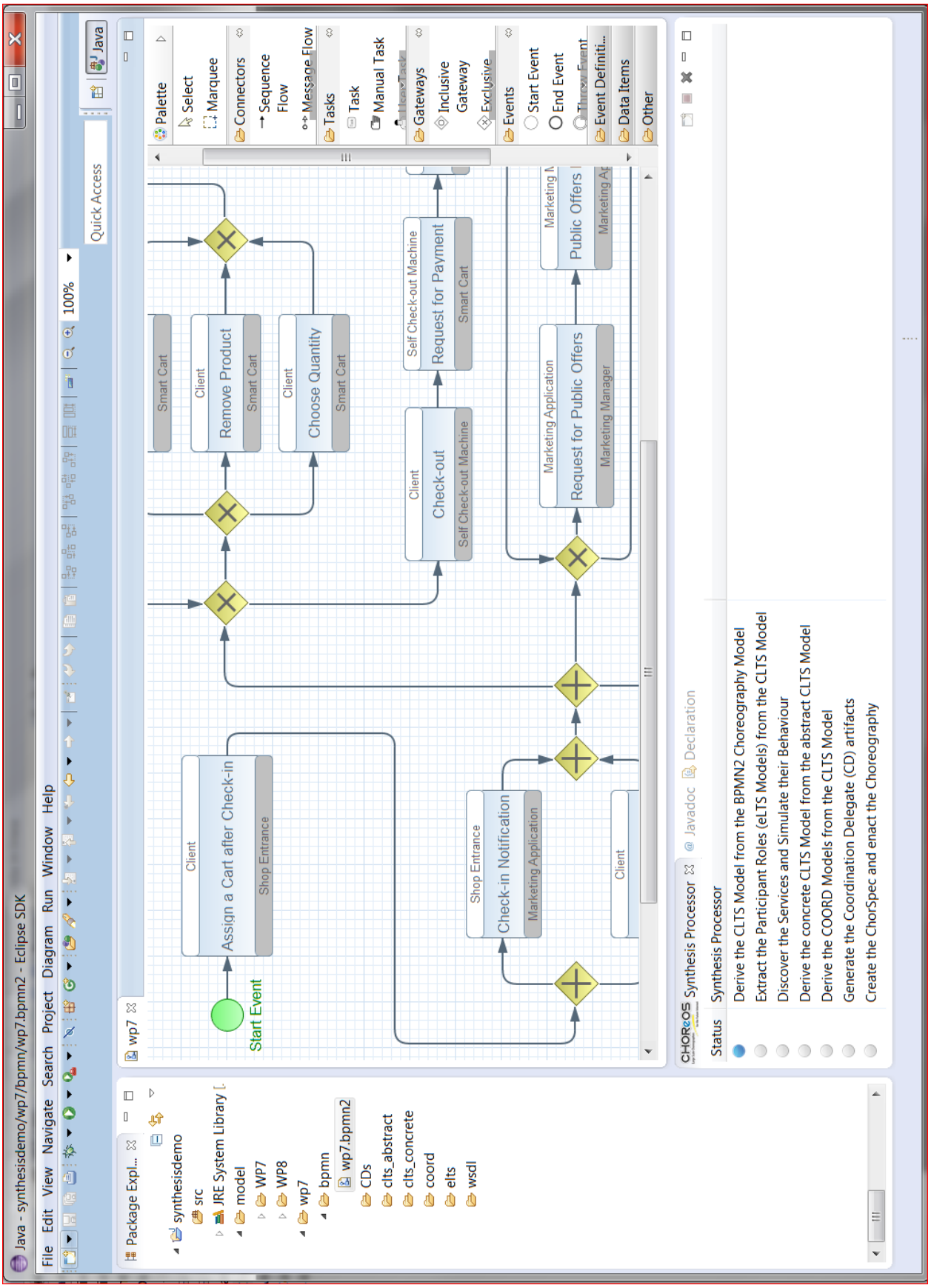


Figure A.3: The BPMN2 model has been loaded by the Synthesis Processor and a complete folder structure has been created to contain all the artefacts that are going to be produced

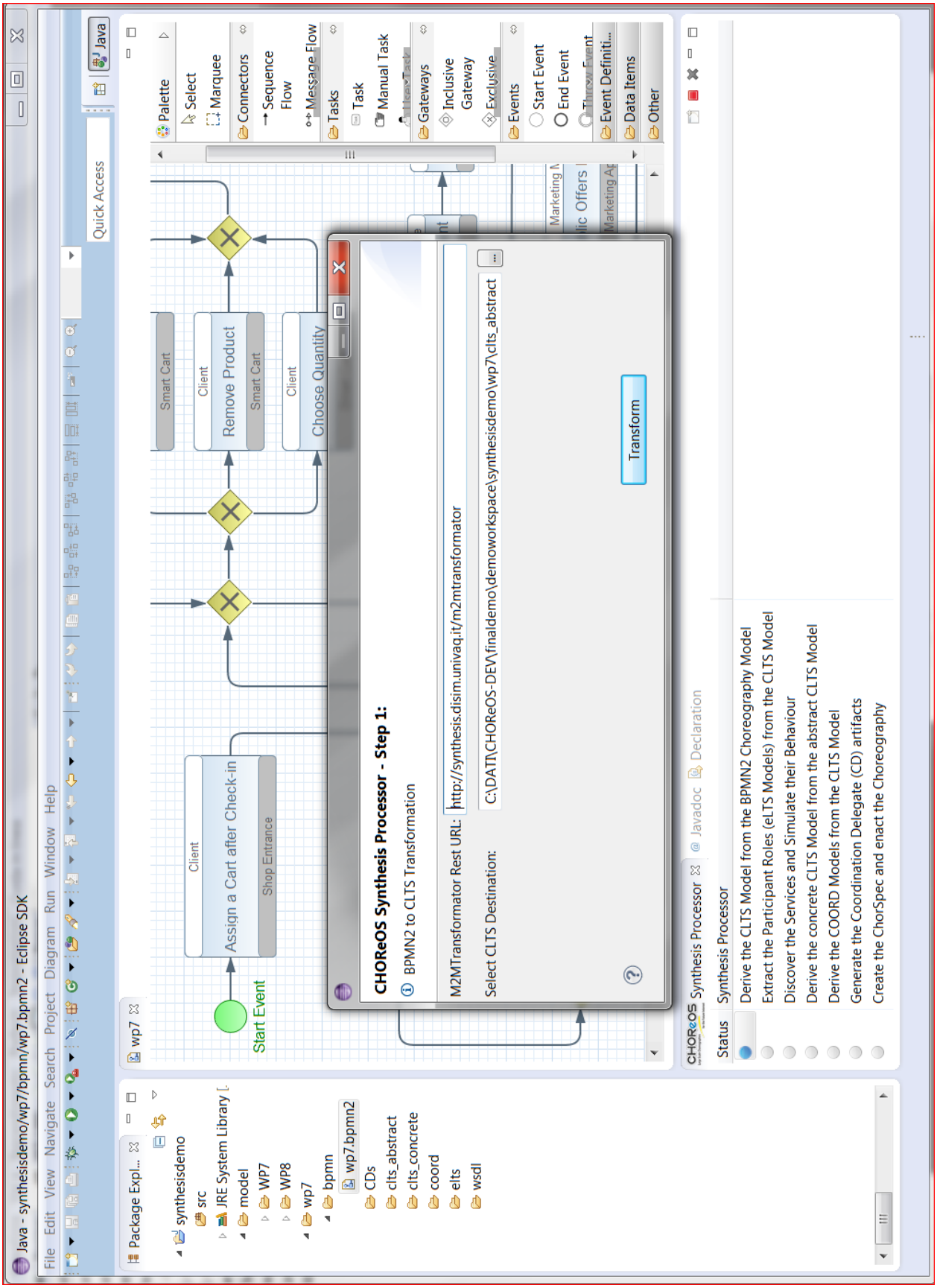


Figure A.4: The REST service `m2mtrasnformator` is going to be called to transform the BPMN2 model to the corresponding CLTS

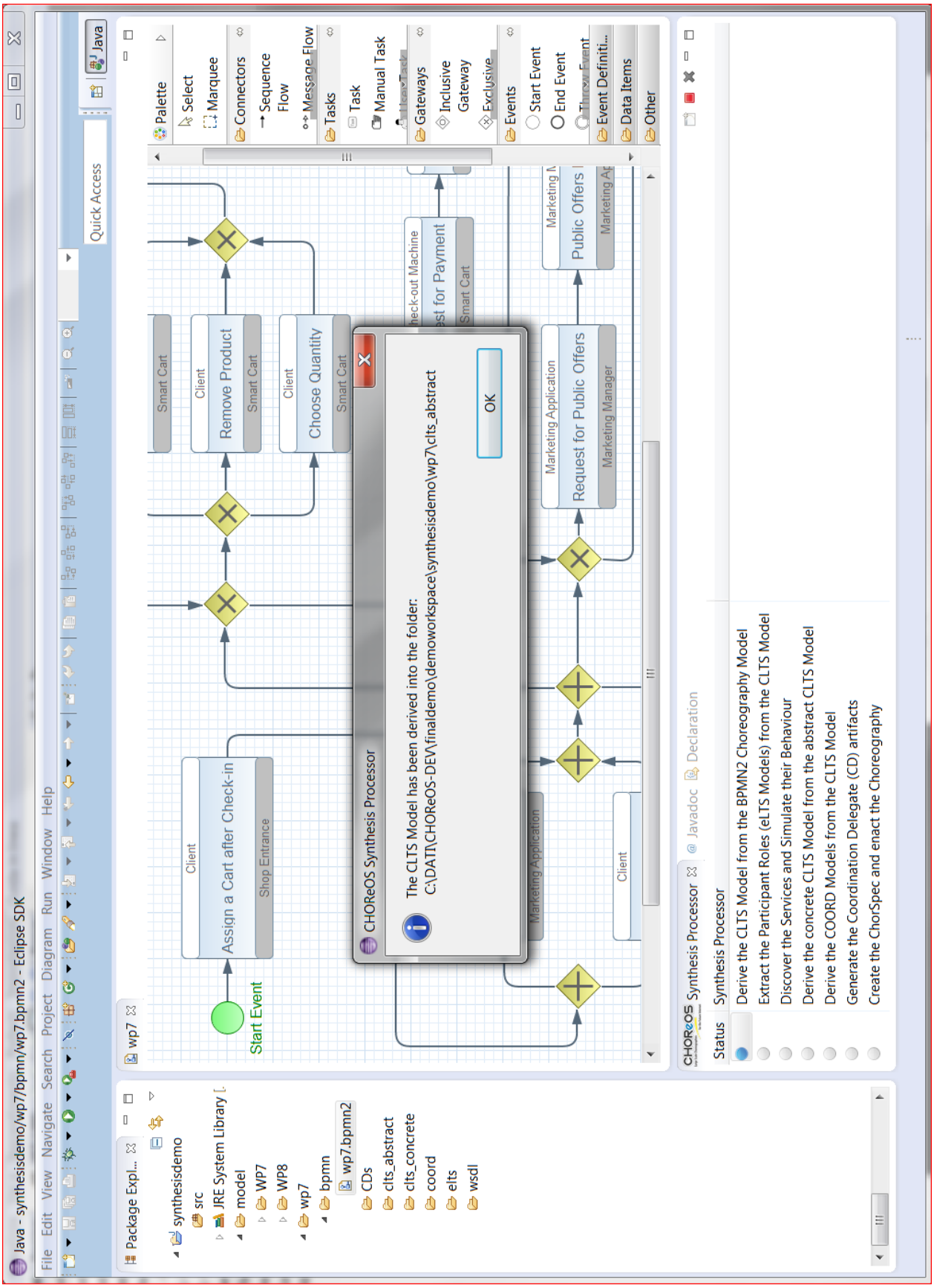


Figure A.5: The CLTS has been derived in to the dedicated folder

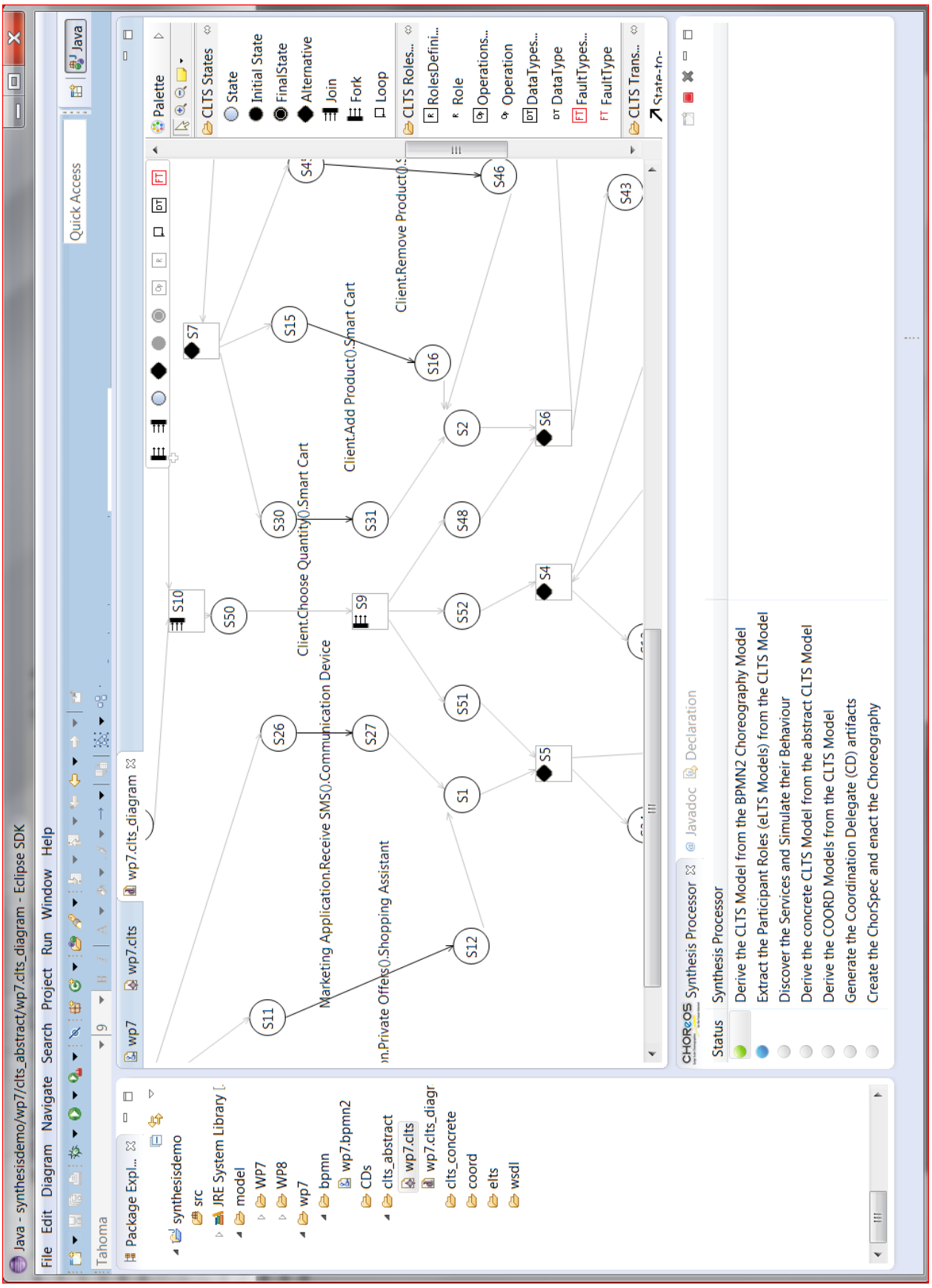


Figure A.6: After initializing the related diagram (through right click on the generated ".clts" file), the CLTS is shown by using the dedicated GMF-based graphical editor

Figure A.7: The generated CLTS is shown by using the native tree view

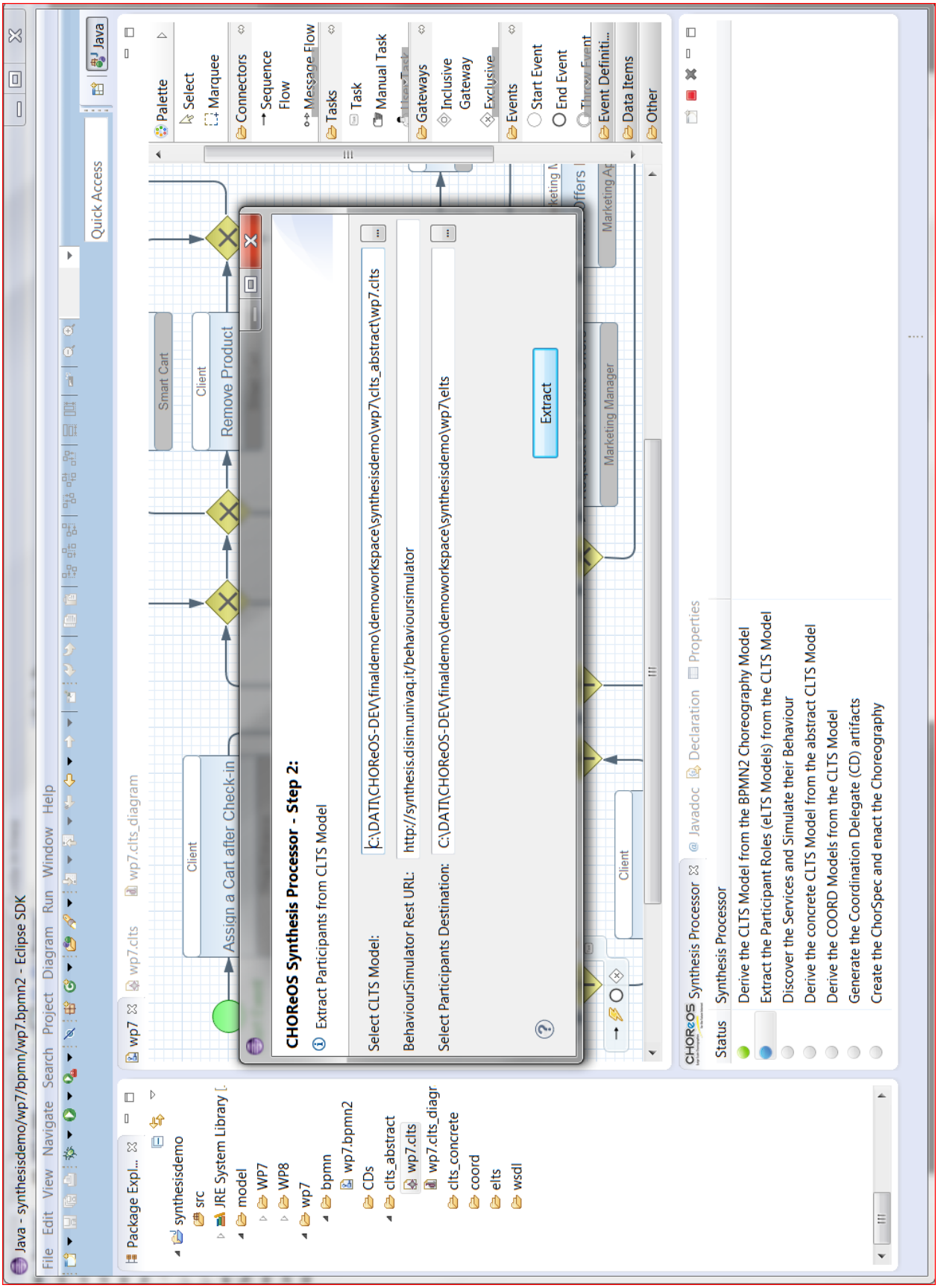


Figure A.8: The REST service `behavioursimulator` is going to be called to project the generated CLTS into a set of eLTS, one for each participant of the choreography. The eLTSs model the expected behaviours that the services to be discovered must simulate so to be able to play the roles of the participants

Figure A.9: All the eLTSs have been derived in to the dedicated folder “elts”

Figure A.10: The folder “elts” containing all the generated eLTS is shown

Figure A.11: The REST service `behavioursimulator` is going to be called once more to interact with the eXtensible Service Discovery (XSD), and in particular with the Service Base Query Engine (see Chapter 3), hence (possibly) discovering services whose behaviour simulate the roles of the participants (i.e., the eLTS)

Figure A.12: All the needed services have been found and, hence, all the participants roles can be played. Whenever the Query Engine should not be able to discover some services, the process is stopped and the user is alerted

Figure A.13: All the WSDLs of the discovered services are downloaded into the dedicated folder

Figure A.14: Considering the WSDLs of the discovered services, the REST service `m2mtrasnformator` is going to be called once more to transform the abstract CLTS to the concrete CLTS, where the actual names of the discovered services, as well as, the full qualifiers of the actual operations are used

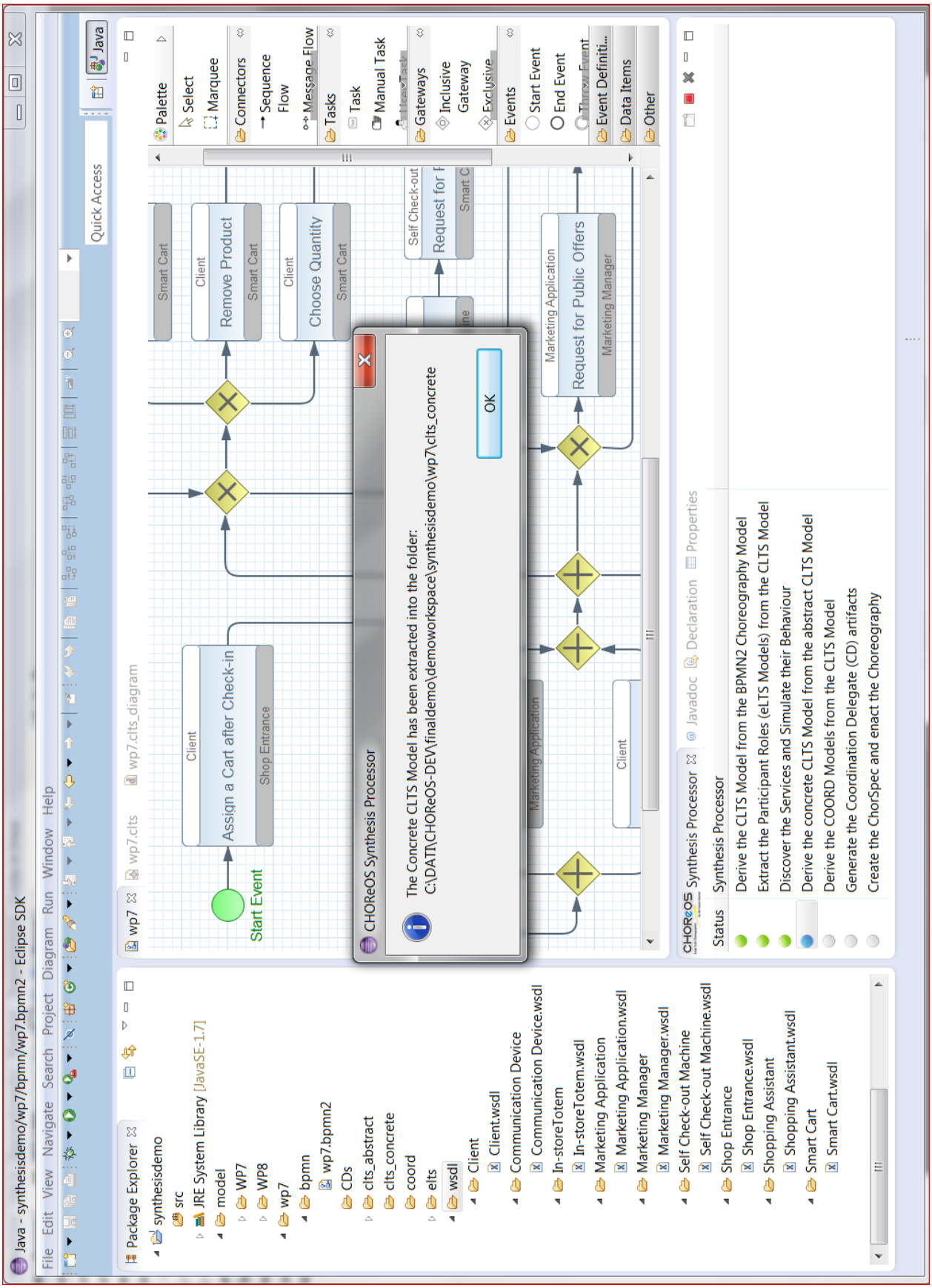


Figure A.15: The concrete CLTS has been generated into the dedicated folder

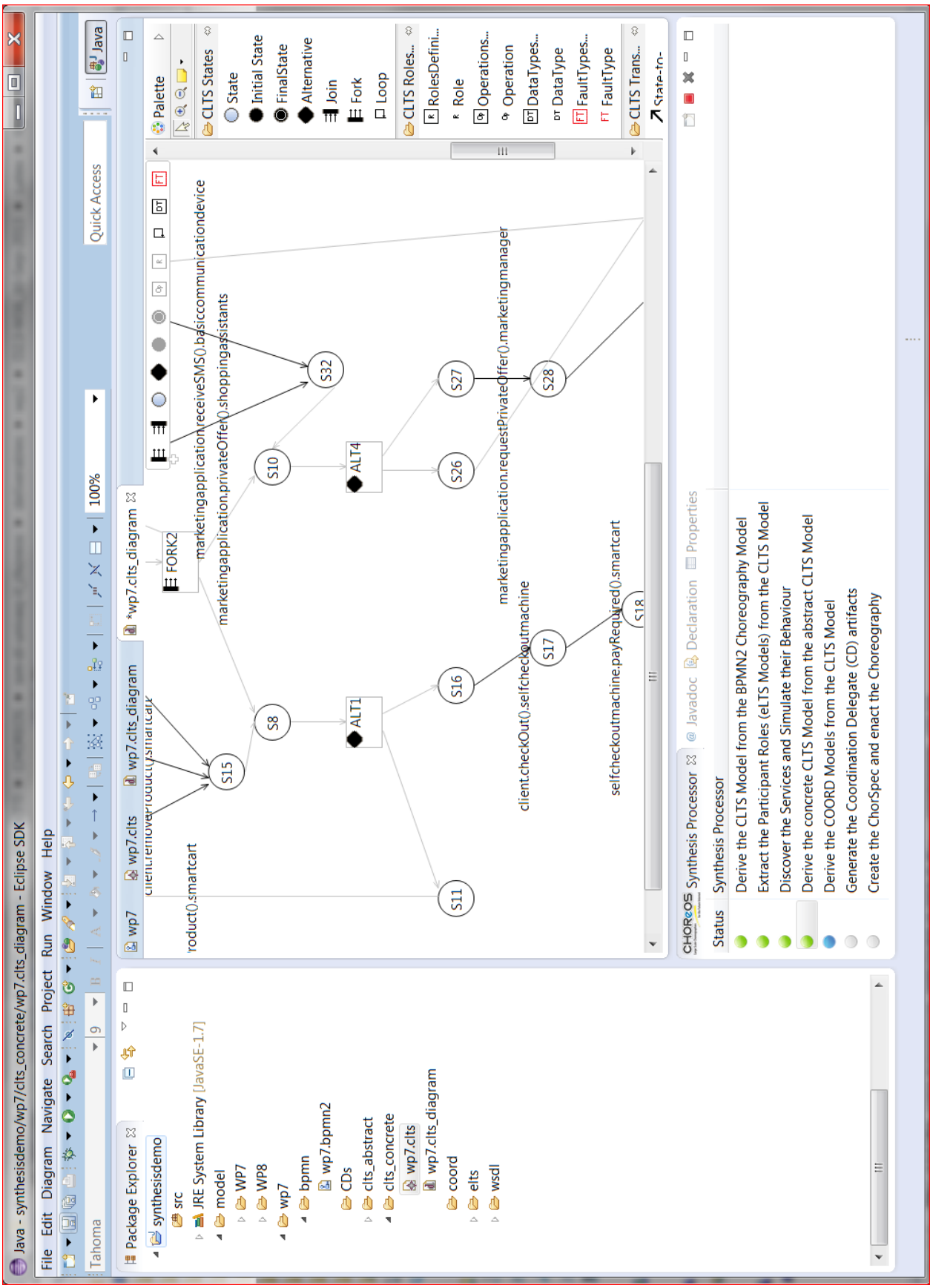


Figure A.16: After initializing the related diagram (through right click on the generated ".clts" file), the concrete CLTS is shown by using the dedicated GMF-based graphical editor

Figure A.17: The REST service `m2mtrasnformator` is going to be called once more to generate all the Coord Models out of the concrete CLTS

Figure A.18: The Coord Models have been generated into the dedicated folder “coord”

Figure A.19: All the Coord Models are shown into the folder “coord” on the left side, and the tree view of the coordination tuples for one of them is shown on the right side

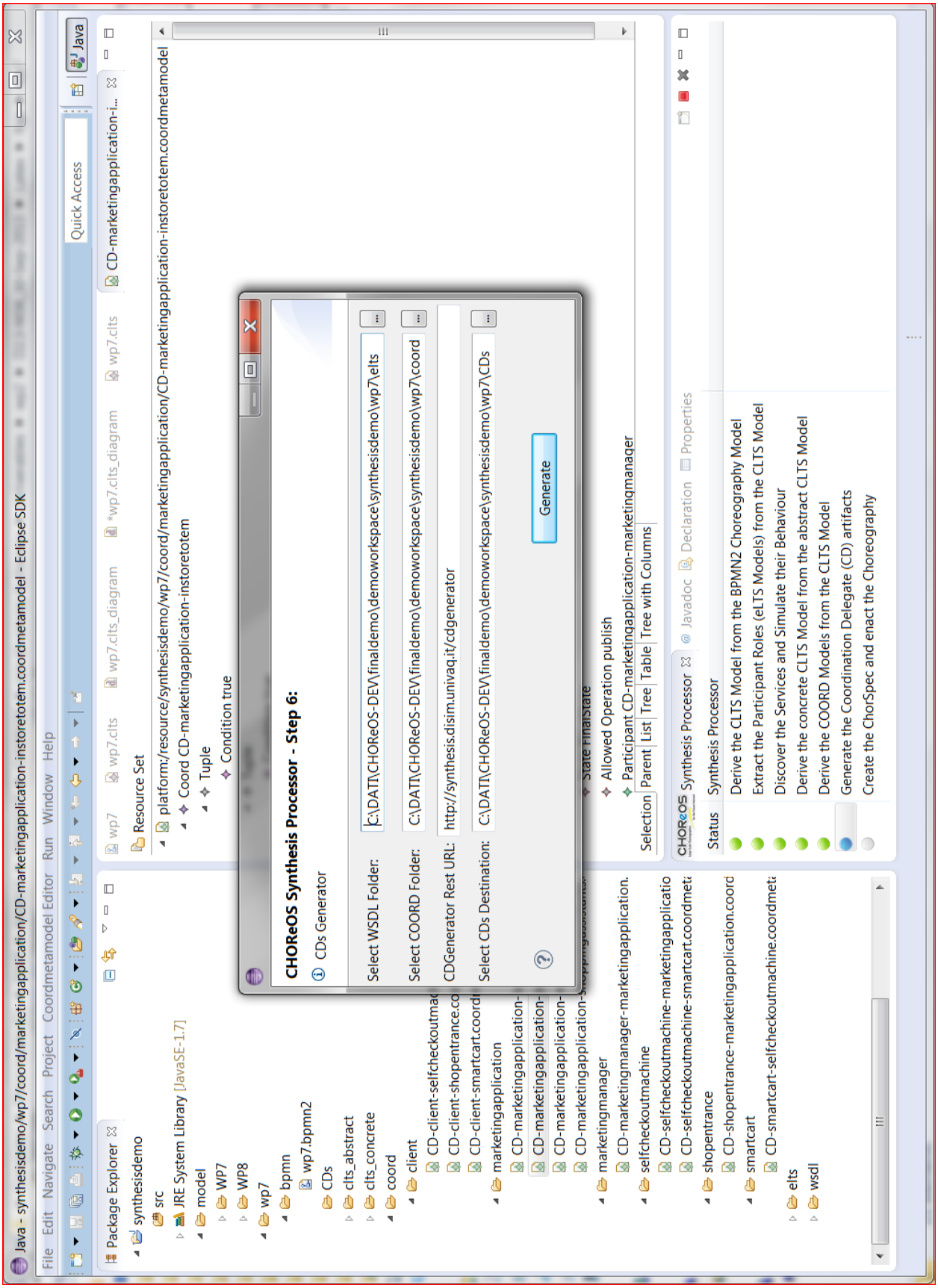


Figure A.20: The REST service `cdgenerator` is going to be called to generated all the Coordination Delegates (CDs)

Figure A.21: The Coordination Delegates have been generated into the dedicated folder

Figure A.22: The Coordination Delegates (CDs) are shown into the dedicated folder

Figure A.23: The `ChorSpec` of the overall choreography is going to be generated

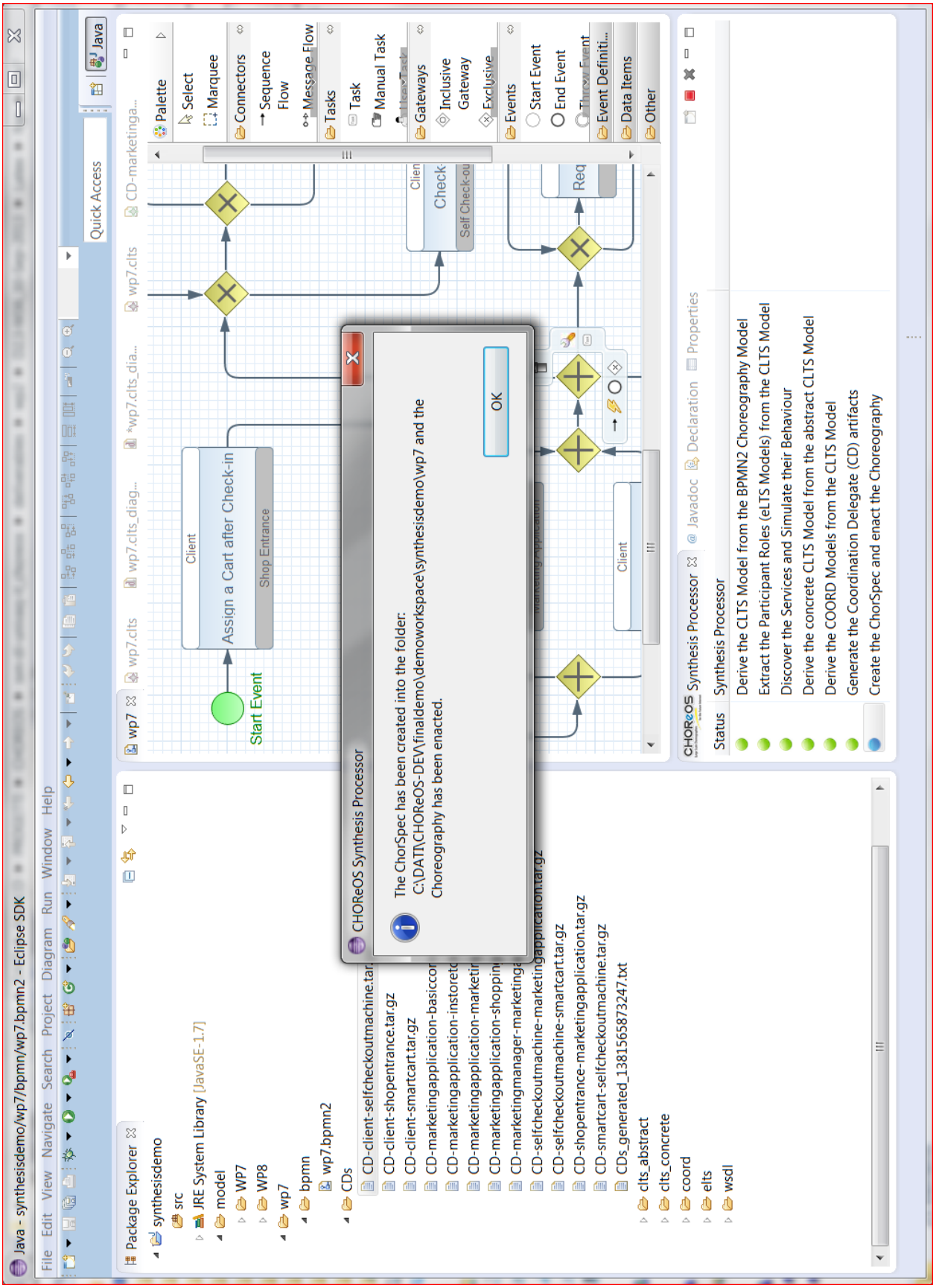


Figure A.24: The `ChorSpec` has been created into the dedicated folder and it is passed to the Enactment Engine (EE) to enact the choreography

Figure A.25: An excerpt of the XML file codifying the `ChorSpec` is shown. The overall XML file is reported below in Listing A.1

Figure A.26: To the convenience of the CHOReOS synthesis user, a text file is also generated reporting the list of all the CDs that have been generated and the URLs from where they can be downloaded

Figure A.27: A CD is unpackaged to show the artefacts it is made of (left-hand bottom side): a configuration file, a property file, an XMI file containing its coordination tuples, and the WSDL of the proxyfied service

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<choreography>
    <choreographySpec>
        <deployableServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-smartcart</serviceSpecName>
                <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-shopentrance</serviceSpecName>
                <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <name>CD-client-selfcheckoutmachine</name>
            <roles>selfcheckoutmachine</roles>
            <serviceType>
                <type>COORDEL</type>
            </serviceType>
            <endpointName>CD-client-selfcheckoutmachine</endpointName>
            <numberOfInstances>1</numberOfInstances>
            <packageType>
                <type>EASY_ESB</type>
            </packageType>
            <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146272/CD-client-
                selfcheckoutmachine.tar.gz</packageUri>
            <port>8180</port>
        </deployableServiceSpecs>
        <deployableServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
```

```
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
            <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>shopentrance</serviceSpecName>
            <serviceSpecRole>shopentrance</serviceSpecRole>
        </dependencies>
        <name>CD-client-shopentrance</name>
        <roles>shopentrance</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-client-shopentrance</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146314/CD-client-
            shopentrance.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <deployableServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
        </dependencies>
```

```
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>smartcart</serviceSpecName>
            <serviceSpecRole>smartcart</serviceSpecRole>
        </dependencies>
        <name>CD-client-smartcart</name>
        <roles>smartcart</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-client-smartcart</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146363/CD-client-
            smartcart.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <deployableServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
```

```
            <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>marketingapplication</serviceSpecName>
            <serviceSpecRole>marketingapplication</serviceSpecRole>
        </dependencies>
        <name>CD-shopentrance-marketingapplication</name>
        <roles>marketingapplication</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-shopentrance-marketingapplication</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146436/CD-
            shopentrance-marketingapplication.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <deployableServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
            <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
```

```
            <dependencies>
                <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-shopentrance</serviceSpecName>
                <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>instoretotem</serviceSpecName>
                <serviceSpecRole>instoretotem</serviceSpecRole>
            </dependencies>
            <name>CD-marketingapplication-instoretotem</name>
            <roles>instoretotem</roles>
            <serviceType>
                <type>COORDEL</type>
            </serviceType>
            <endpointName>CD-marketingapplication-instoretotem</endpointName>
            <numberOfInstances>1</numberOfInstances>
            <packageType>
                <type>EASY_ESB</type>
            </packageType>
            <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146476/CD-
                marketingapplication-instoretotem.tar.gz</packageUri>
            <port>8180</port>
        </deployableServiceSpecs>
        <deployableServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-smartcart</serviceSpecName>
                <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-shopentrance</serviceSpecName>
                <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>shoppingassistants</serviceSpecName>
```

```
            <serviceSpecRole>shoppingassistants</serviceSpecRole>
        </dependencies>
        <name>CD-marketingapplication-shoppingassistants</name>
        <roles>shoppingassistants</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-marketingapplication-shoppingassistants</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146528/CD-
            marketingapplication-shoppingassistants.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <deployableServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
            <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>marketingmanager</serviceSpecName>
            <serviceSpecRole>marketingmanager</serviceSpecRole>
        </dependencies>
        <name>CD-marketingapplication-marketingmanager</name>
        <roles>marketingmanager</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-marketingapplication-marketingmanager</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
```

```
433             <type>EASY_ESB</type>
434          </packageType>
435          <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146570/CD-
                 marketingapplication-marketingmanager.tar.gz</packageUri>
436          <port>8180</port>
437       </deployableServiceSpecs>
438       <deployableServiceSpecs>
439          <dependencies>
440             <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
441             <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
442          </dependencies>
443          <dependencies>
444             <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
445             <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
446          </dependencies>
447          <dependencies>
448             <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
449             <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
450          </dependencies>
451          <dependencies>
452             <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
453             <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
454          </dependencies>
455          <dependencies>
456             <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
457             <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
458          </dependencies>
459          <dependencies>
460             <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
461             <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
462          </dependencies>
463          <dependencies>
464             <serviceSpecName>CD-client-smartcart</serviceSpecName>
465             <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
466          </dependencies>
467          <dependencies>
468             <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
469             <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
470          </dependencies>
471          <dependencies>
472             <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
473             <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
474          </dependencies>
475          <dependencies>
476             <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
477             <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
478          </dependencies>
479          <dependencies>
480             <serviceSpecName>CD-client-shopentrance</serviceSpecName>
481             <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
482          </dependencies>
483          <dependencies>
484             <serviceSpecName>basiccommunicationdevice</serviceSpecName>
485             <serviceSpecRole>basiccommunicationdevice</serviceSpecRole>
486          </dependencies>
487          <name>CD-marketingapplication-basiccommunicationdevice</name>
488          <roles>basiccommunicationdevice</roles>
489          <serviceType>
490             <type>COORDEL</type>
491          </serviceType>
492          <endpointName>CD-marketingapplication-basiccommunicationdevice</endpointName>
493          <numberOfInstances>1</numberOfInstances>
494          <packageType>
495             <type>EASY_ESB</type>
496          </packageType>
497          <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146612/CD-
                 marketingapplication-basiccommunicationdevice.tar.gz</packageUri>
498          <port>8180</port>
499       </deployableServiceSpecs>
500       <deployableServiceSpecs>
501          <dependencies>
502             <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
503             <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
```

```
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-smartcart</serviceSpecName>
                <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-shopentrance</serviceSpecName>
                <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>marketingapplication</serviceSpecName>
                <serviceSpecRole>marketingapplication</serviceSpecRole>
            </dependencies>
            <name>CD-selfcheckoutmachine-marketingapplication</name>
            <roles>marketingapplication</roles>
            <serviceType>
                <type>COORDEL</type>
            </serviceType>
            <endpointName>CD-selfcheckoutmachine-marketingapplication</endpointName>
            <numberOfInstances>1</numberOfInstances>
            <packageType>
                <type>EASY_ESB</type>
            </packageType>
            <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146675/CD-
                selfcheckoutmachine-marketingapplication.tar.gz</packageUri>
            <port>8180</port>
        </deployableServiceSpecs>
        <deployableServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
            </dependencies>
            <dependencies>
```

```
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
            <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>smartcart</serviceSpecName>
            <serviceSpecRole>smartcart</serviceSpecRole>
        </dependencies>
        <name>CD-selfcheckoutmachine-smartcart</name>
        <roles>smartcart</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-selfcheckoutmachine-smartcart</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146725/CD-
            selfcheckoutmachine-smartcart.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <deployableServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
            <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
```

```
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-smartcart</serviceSpecName>
                <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-smartcart-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-shopentrance</serviceSpecName>
                <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>marketingapplication</serviceSpecName>
                <serviceSpecRole>marketingapplication</serviceSpecRole>
            </dependencies>
            <name>CD-marketingmanager-marketingapplication</name>
            <roles>marketingapplication</roles>
            <serviceType>
                <type>COORDEL</type>
            </serviceType>
            <endpointName>CD-marketingmanager-marketingapplication</endpointName>
            <numberOfInstances>1</numberOfInstances>
            <packageType>
                <type>EASY_ESB</type>
            </packageType>
            <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146798/CD-
                marketingmanager-marketingapplication.tar.gz</packageUri>
            <port>8180</port>
        </deployableServiceSpecs>
        <deployableServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-marketingmanager-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
                <serviceSpecRole>CD-selfcheckoutmachine-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-marketingmanager</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>CD-client-selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
                <serviceSpecRole>CD-shopentrance-marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
                <serviceSpecRole>CD-marketingapplication-basiccommunicationdevice</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-client-smartcart</serviceSpecName>
                <serviceSpecRole>CD-client-smartcart</serviceSpecRole>
            </dependencies>
            <dependencies>
```

```
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>CD-marketingapplication-instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>CD-client-shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <name>CD-smartcart-selfcheckoutmachine</name>
        <roles>selfcheckoutmachine</roles>
        <serviceType>
            <type>COORDEL</type>
        </serviceType>
        <endpointName>CD-smartcart-selfcheckoutmachine</endpointName>
        <numberOfInstances>1</numberOfInstances>
        <packageType>
            <type>EASY_ESB</type>
        </packageType>
        <packageUri>http://synthesis.disim.univaq.it/cdgenerator/getcd/wp7-soap/1382102146848/CD-smartcart-
            selfcheckoutmachine.tar.gz</packageUri>
        <port>8180</port>
    </deployableServiceSpecs>
    <legacyServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-client-selfcheckoutmachine</serviceSpecName>
            <serviceSpecRole>selfcheckoutmachine</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-shopentrance</serviceSpecName>
            <serviceSpecRole>shopentrance</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-client-smartcart</serviceSpecName>
            <serviceSpecRole>smartcart</serviceSpecRole>
        </dependencies>
        <name>client</name>
        <roles>client</roles>
        <serviceType>
            <type>SOAP</type>
        </serviceType>
        <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/client/client?wsdl</nativeURIs>
    </legacyServiceSpecs>
    <legacyServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-shopentrance-marketingapplication</serviceSpecName>
            <serviceSpecRole>marketingapplication</serviceSpecRole>
        </dependencies>
        <name>shopentrance</name>
        <roles>shopentrance</roles>
        <serviceType>
            <type>SOAP</type>
        </serviceType>
        <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/shopentrance/shopentrance?wsdl</nativeURIs>
    </legacyServiceSpecs>
    <legacyServiceSpecs>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-instoretotem</serviceSpecName>
            <serviceSpecRole>instoretotem</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-shoppingassistants</serviceSpecName>
            <serviceSpecRole>shoppingassistants</serviceSpecRole>
        </dependencies>
        <dependencies>
            <serviceSpecName>CD-marketingapplication-marketingmanager</serviceSpecName>
            <serviceSpecRole>marketingmanager</serviceSpecRole>
```

```
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-marketingapplication-basiccommunicationdevice</serviceSpecName>
                <serviceSpecRole>basiccommunicationdevice</serviceSpecRole>
            </dependencies>
            <name>marketingapplication</name>
            <roles>marketingapplication</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/marketingapplication/marketingapplication?wsdl</
                nativeURIs>
        </legacyServiceSpecs>
        <legacyServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-marketingapplication</serviceSpecName>
                <serviceSpecRole>marketingapplication</serviceSpecRole>
            </dependencies>
            <dependencies>
                <serviceSpecName>CD-selfcheckoutmachine-smartcart</serviceSpecName>
                <serviceSpecRole>smartcart</serviceSpecRole>
            </dependencies>
            <name>selfcheckoutmachine</name>
            <roles>selfcheckoutmachine</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/selfcheckoutmachine/selfcheckoutmachine?wsdl</
                nativeURIs>
        </legacyServiceSpecs>
        <legacyServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-marketingmanager-marketingapplication</serviceSpecName>
                <serviceSpecRole>marketingapplication</serviceSpecRole>
            </dependencies>
            <name>marketingmanager</name>
            <roles>marketingmanager</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/marketingmanager/marketingmanager?wsdl</nativeURIs
                >
        </legacyServiceSpecs>
        <legacyServiceSpecs>
            <dependencies>
                <serviceSpecName>CD-smartcart-selfcheckoutmachine</serviceSpecName>
                <serviceSpecRole>selfcheckoutmachine</serviceSpecRole>
            </dependencies>
            <name>smartcart</name>
            <roles>smartcart</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/smartcart/smartcart?wsdl</nativeURIs>
        </legacyServiceSpecs>
        <legacyServiceSpecs>
            <name>instoretotem</name>
            <roles>instoretotem</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/instoretotem/instoretotem?wsdl</nativeURIs>
        </legacyServiceSpecs>
        <legacyServiceSpecs>
            <name>shoppingassistants</name>
            <roles>shoppingassistants</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/shoppingassistantapp/shoppingassistantapp?wsdl</
                nativeURIs>
        </legacyServiceSpecs>
        <legacyServiceSpecs>
```

```
            <name>basiccommunicationdevice</name>
            <roles>basiccommunicationdevice</roles>
            <serviceType>
                <type>SOAP</type>
            </serviceType>
            <nativeURIs>http://choreosdemo.disim.univaq.it/wp7/basiccommunicationdevice/
                basiccommunicationdevice?wsdl</nativeURIs>
        </legacyServiceSpecs>
    </choreographySpec>
    <id>1</id>
</choreography>
```

Listing A.1: `ChorSpec` of the "In-store Marketing and Sale" scenario of the WP7 use case

A.2. Application of the Synthesis Processor to the “Distributed ad-hoc Social Networking” scenario (WP8)

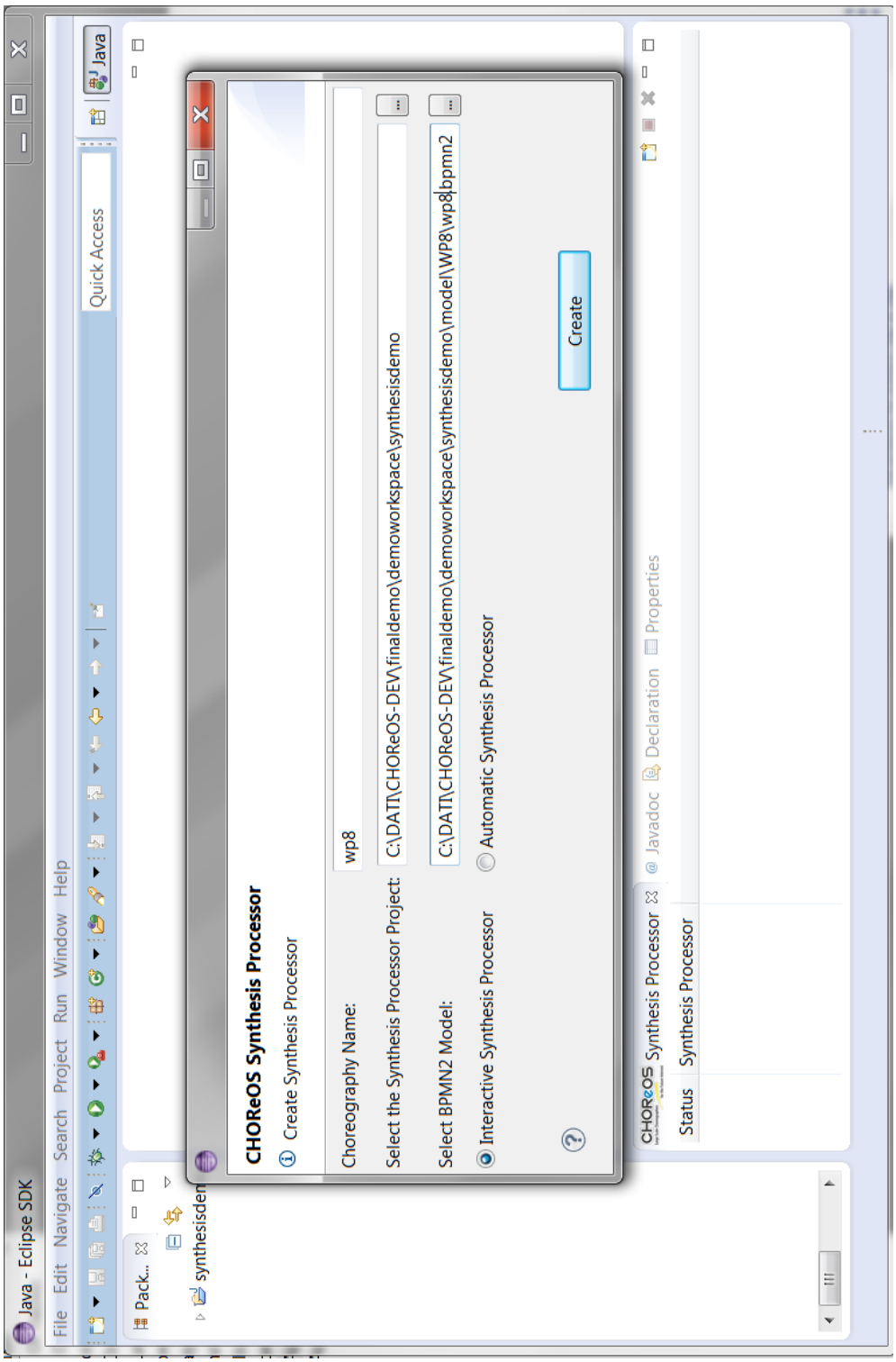


Figure A.28: See the caption of the corresponding figure in the appendix Section A.1

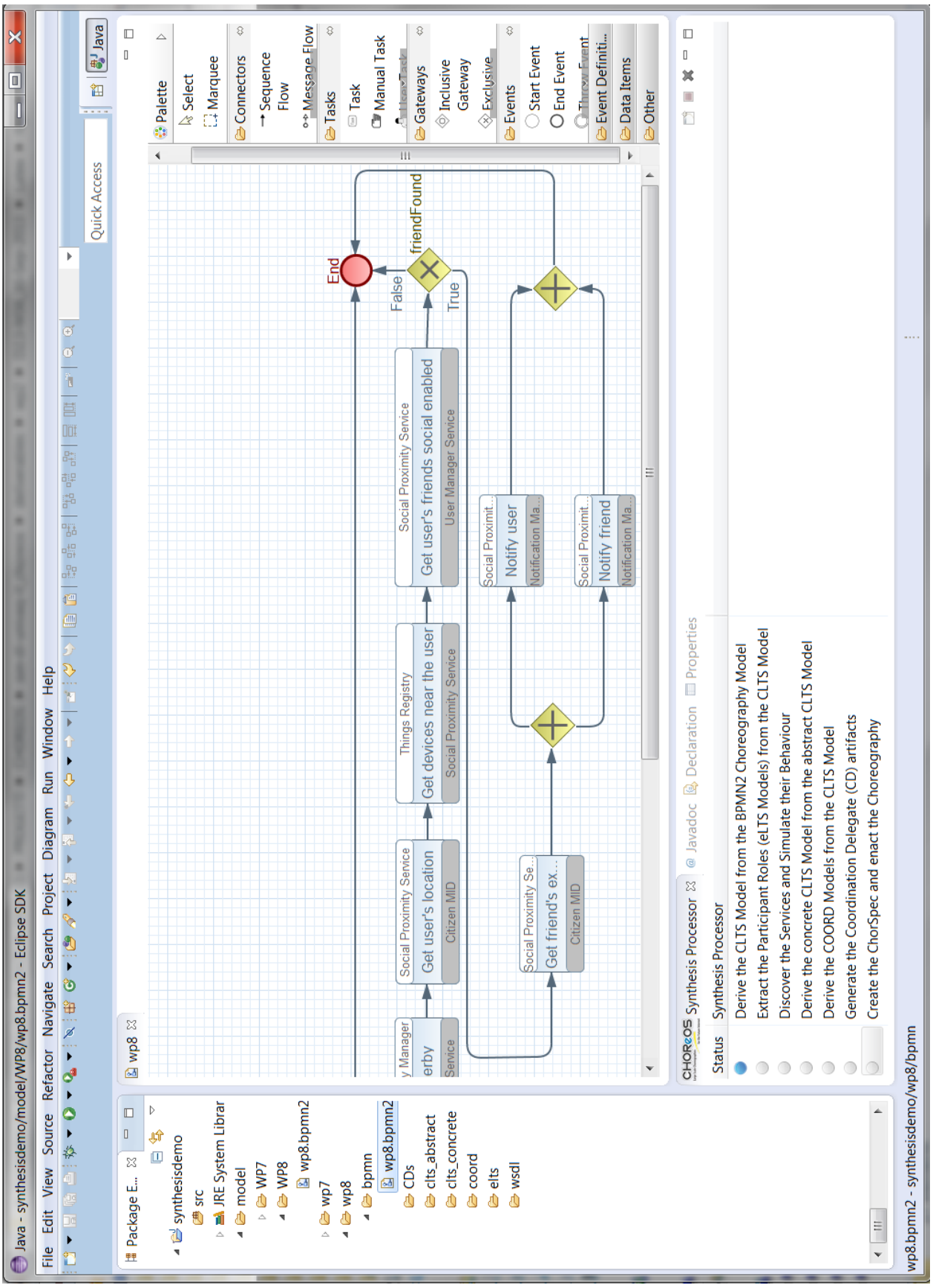


Figure A.29: See the caption of the corresponding figure in the appendix Section A.1

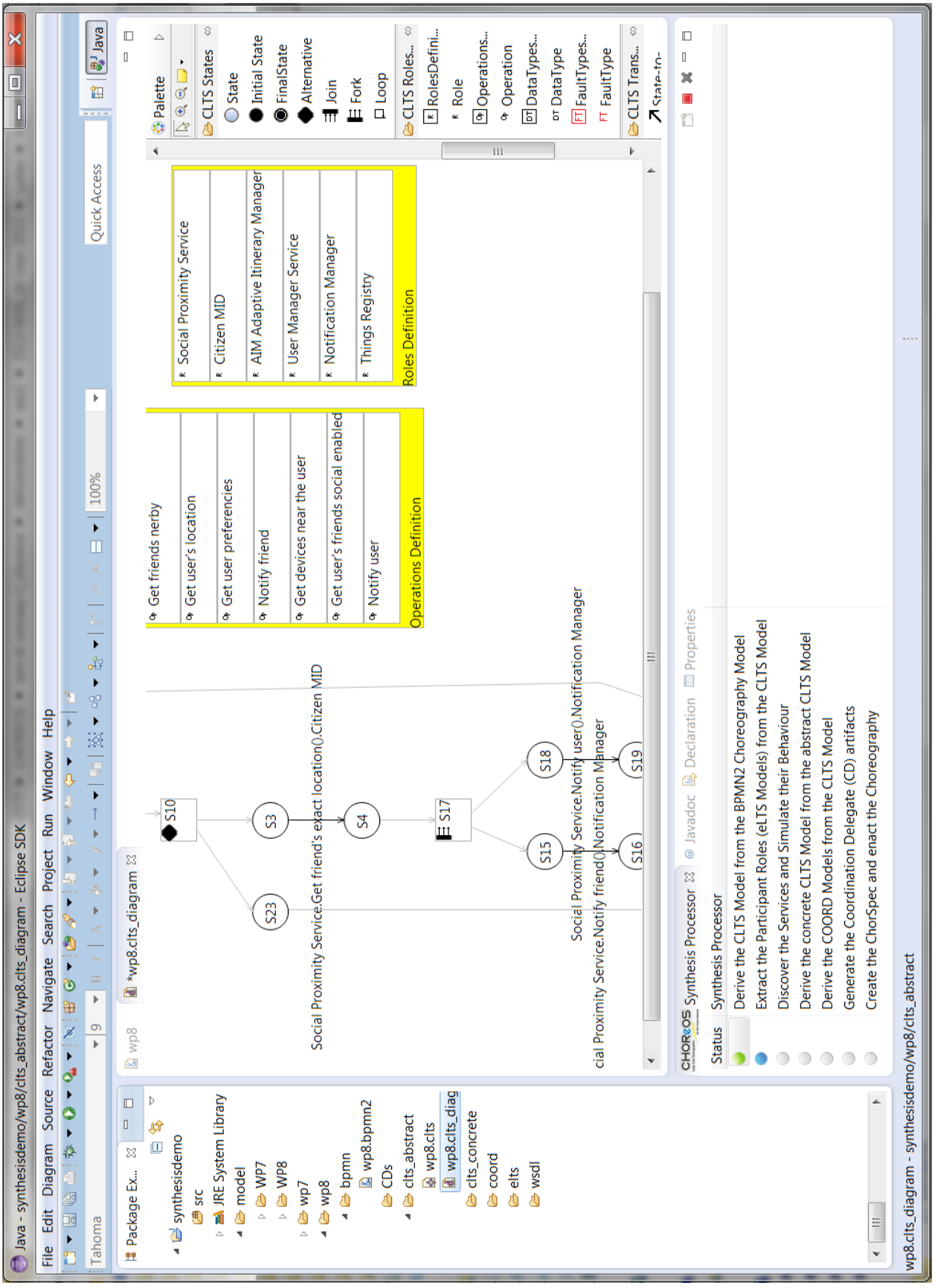


Figure A.30: See the caption of the corresponding figure in the appendix Section A.1

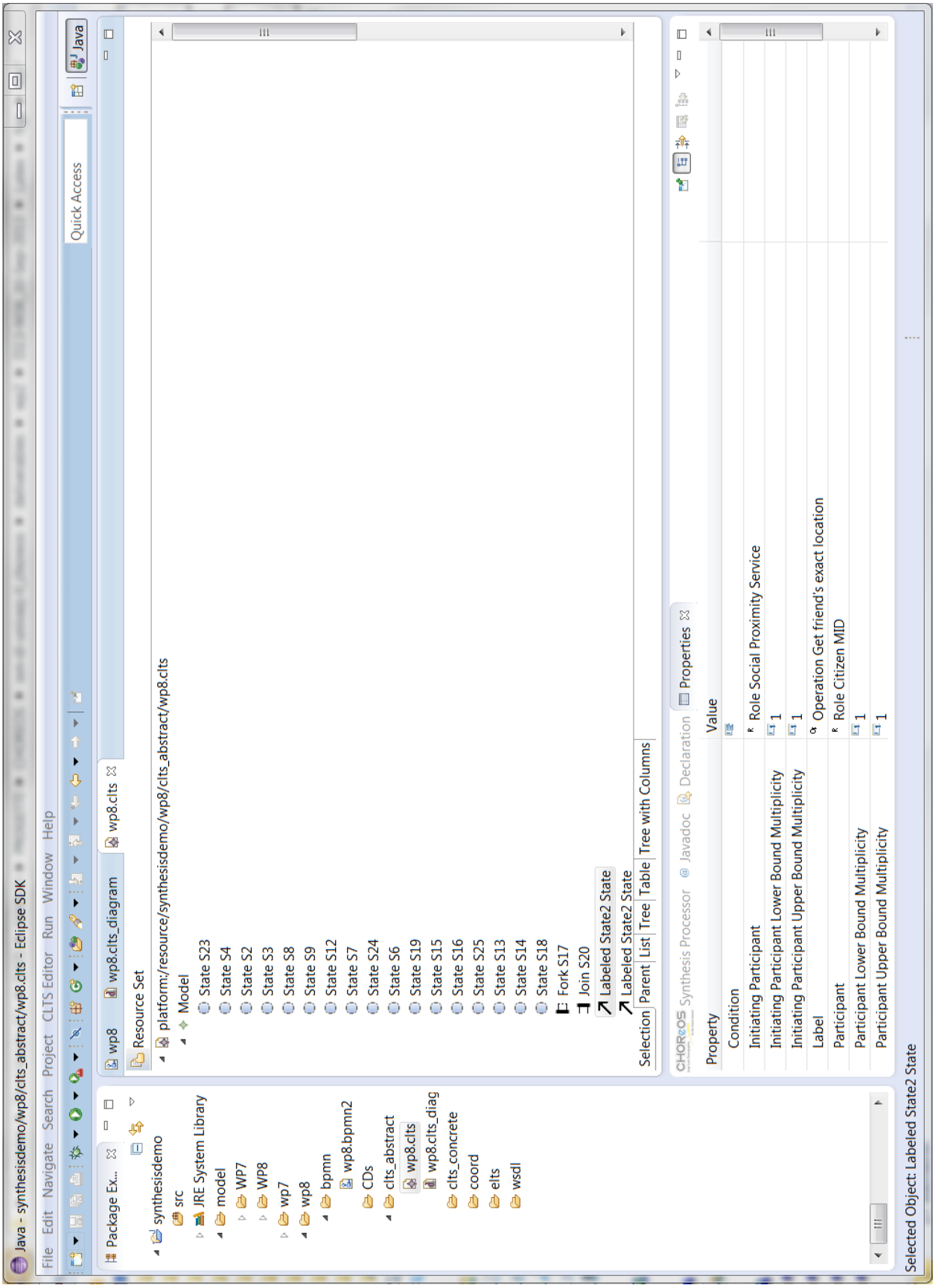


Figure A.31: See the caption of the corresponding figure in the appendix Section A.1

Figure A.32: See the caption of the corresponding figure in the appendix Section A.1

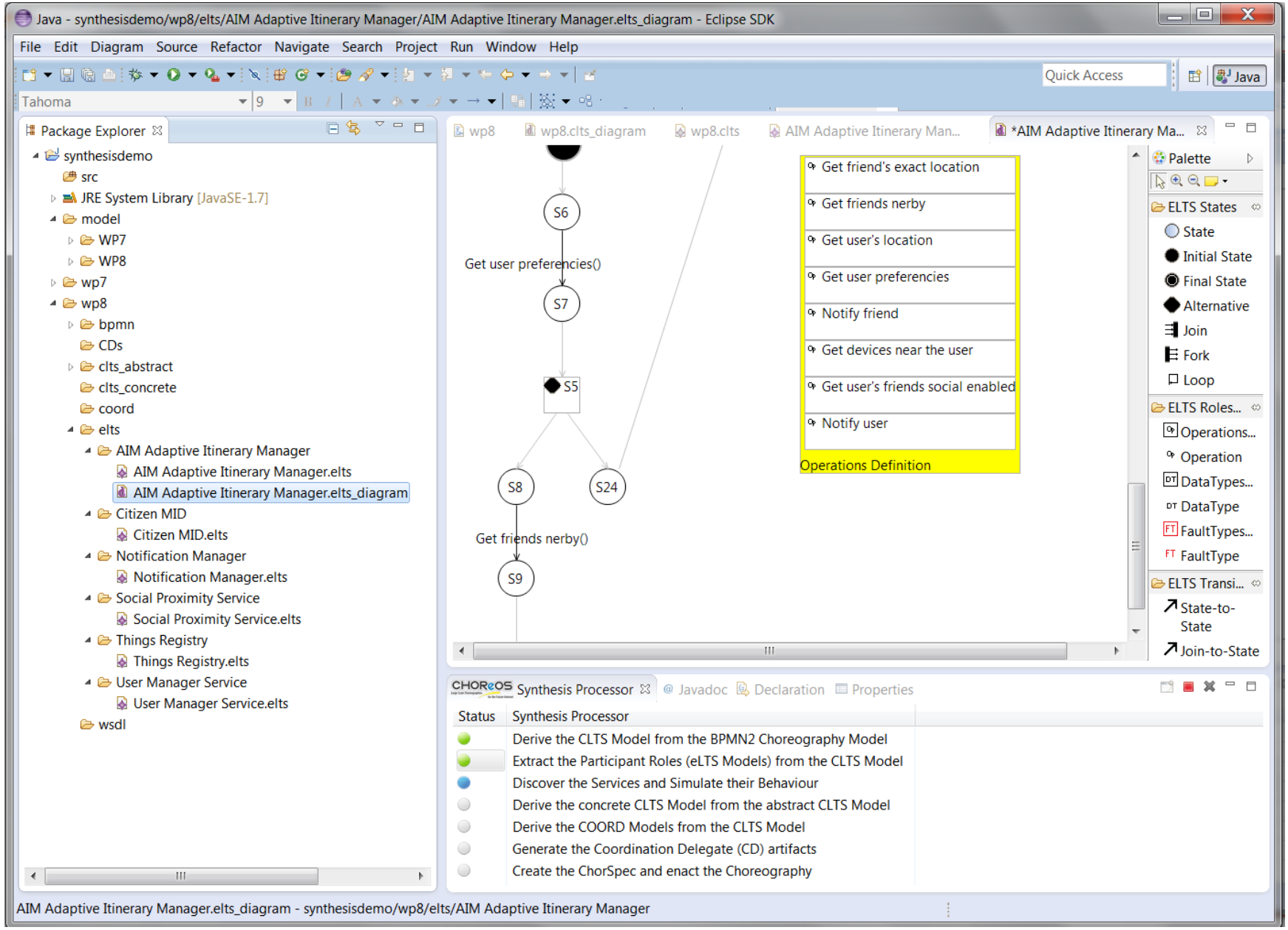


Figure A.33: See the caption of the corresponding figure in the appendix Section A.1

Figure A.34: See the caption of the corresponding figure in the appendix Section A.1

Figure A.35: See the caption of the corresponding figure in the appendix Section A.1

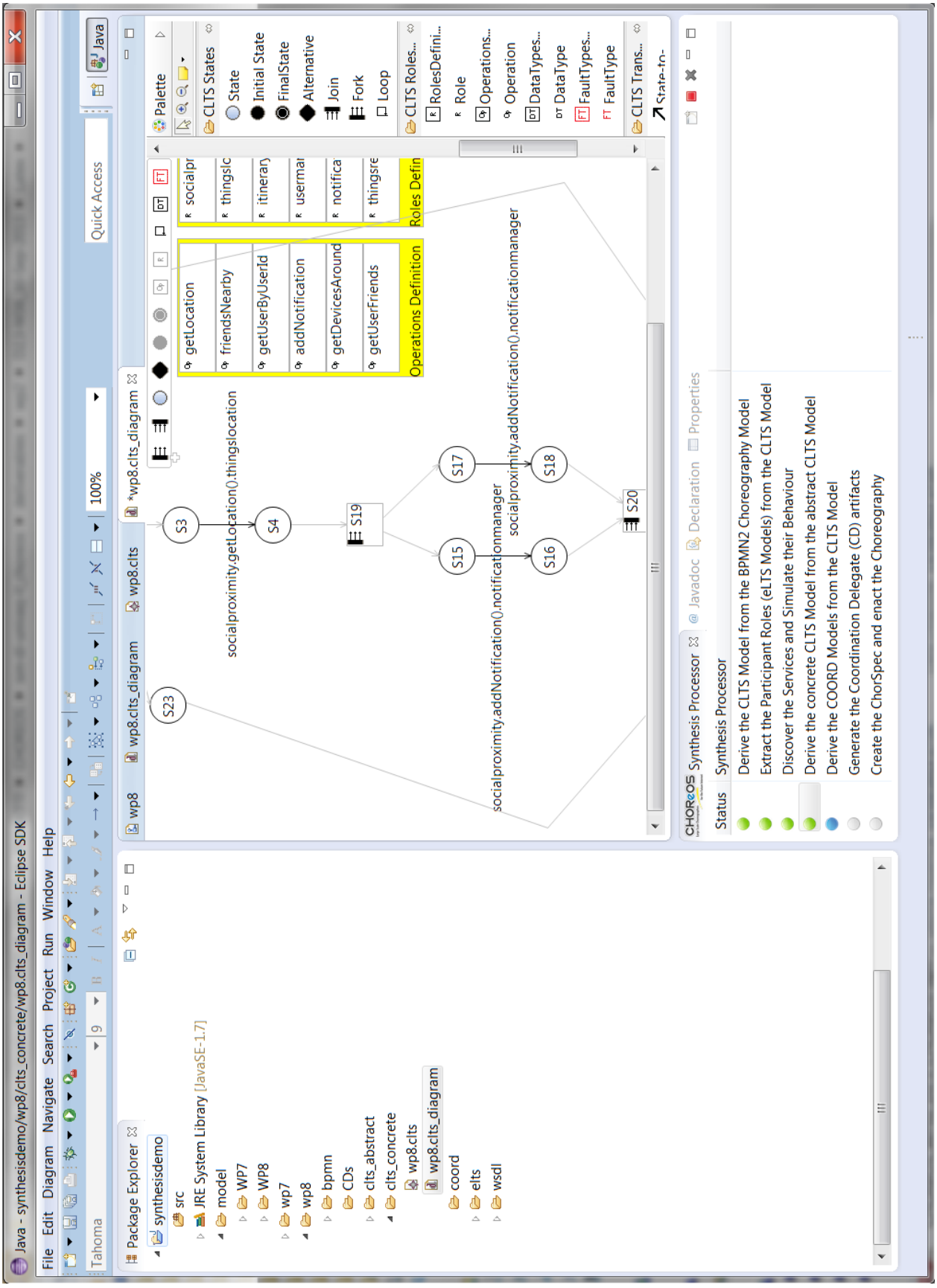


Figure A.36: See the caption of the corresponding figure in the appendix Section A.1

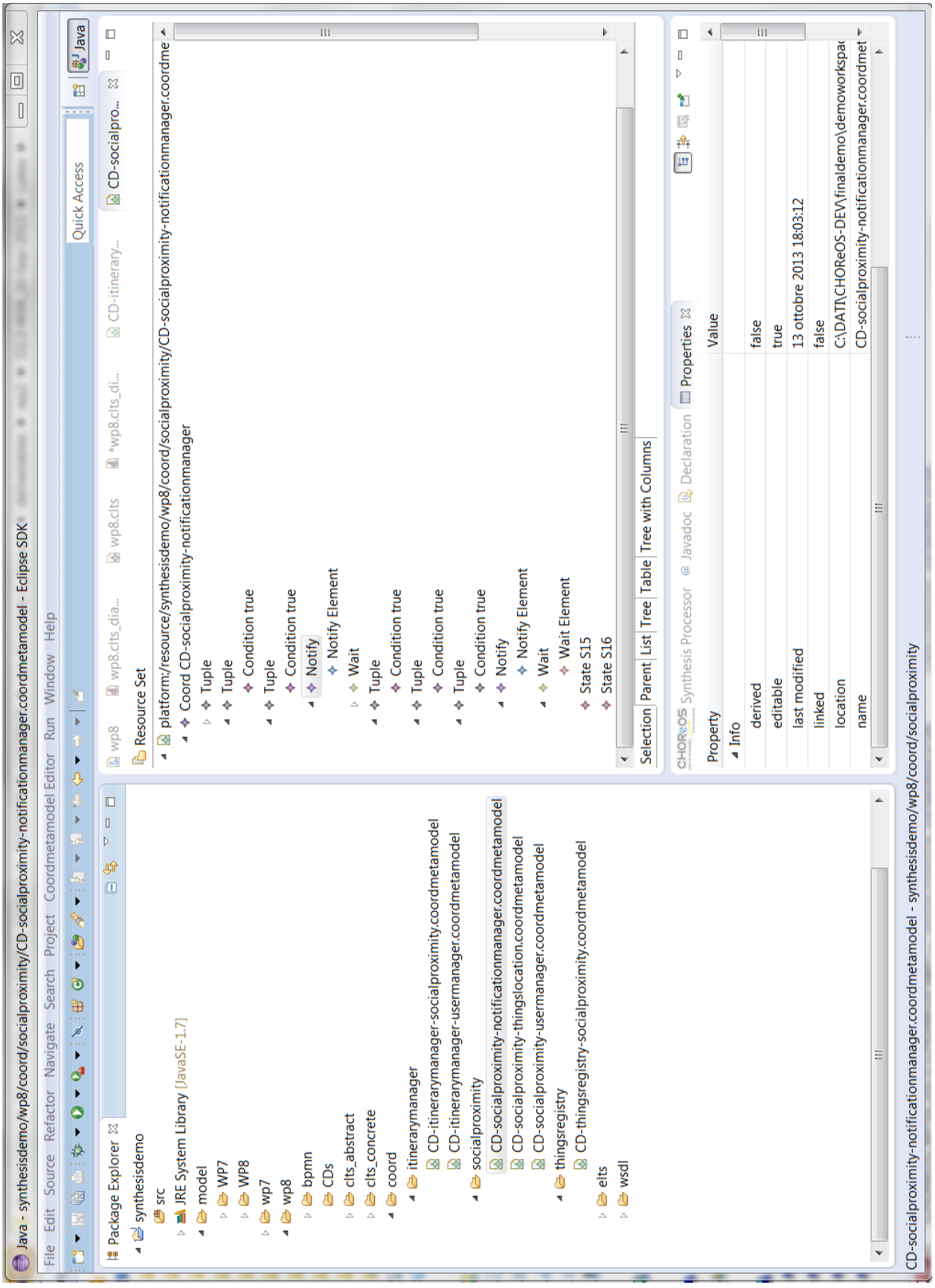


Figure A.37: See the caption of the corresponding figure in the appendix Section A.1

Figure A.38: See the caption of the corresponding figure in the appendix Section A.1

Figure A.39: See the caption of the corresponding figure in the appendix Section A.1

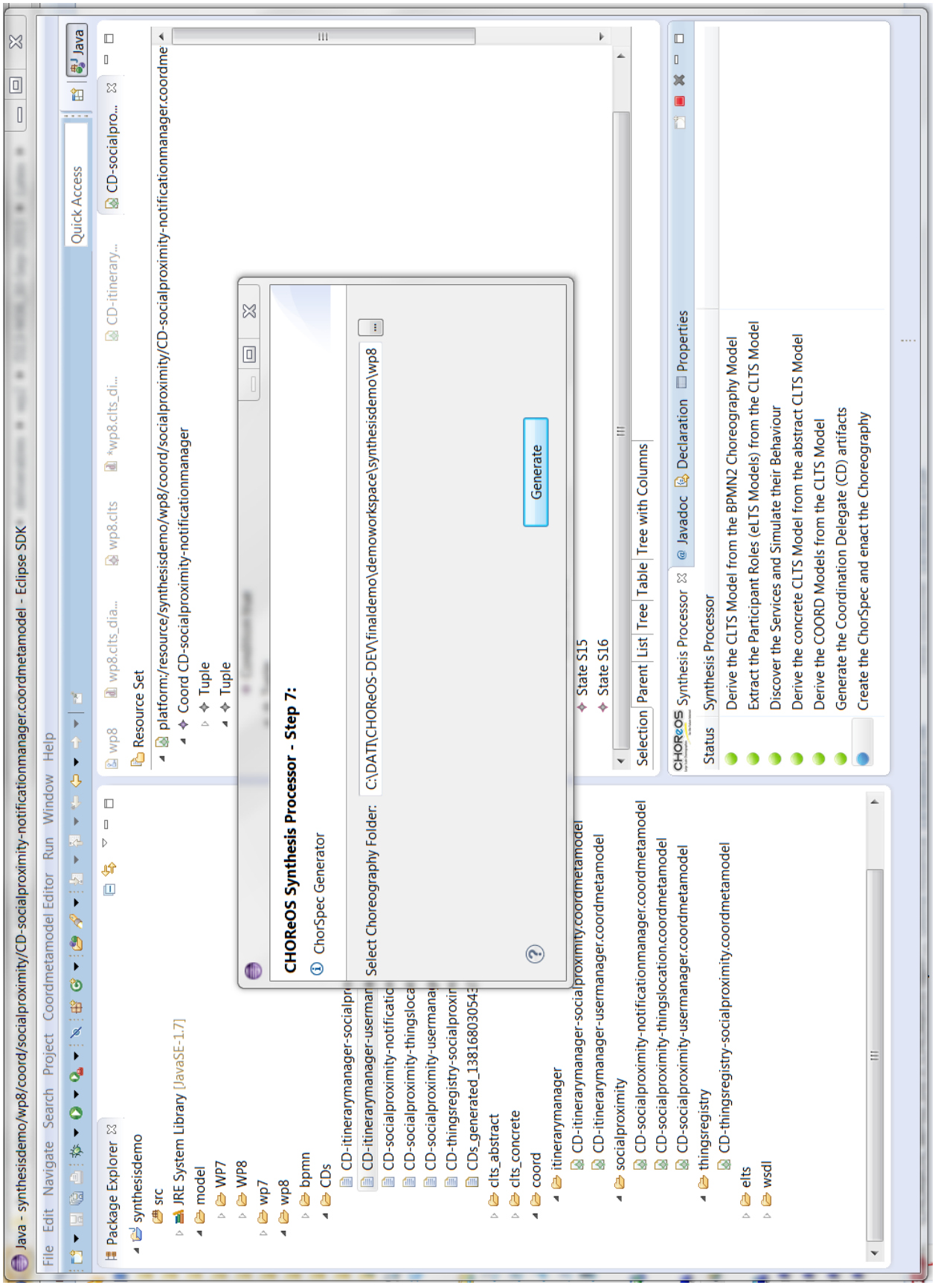


Figure A.40: See the caption of the corresponding figure in the appendix Section A.1

Figure A.41: See the caption of the corresponding figure in the appendix Section A.1

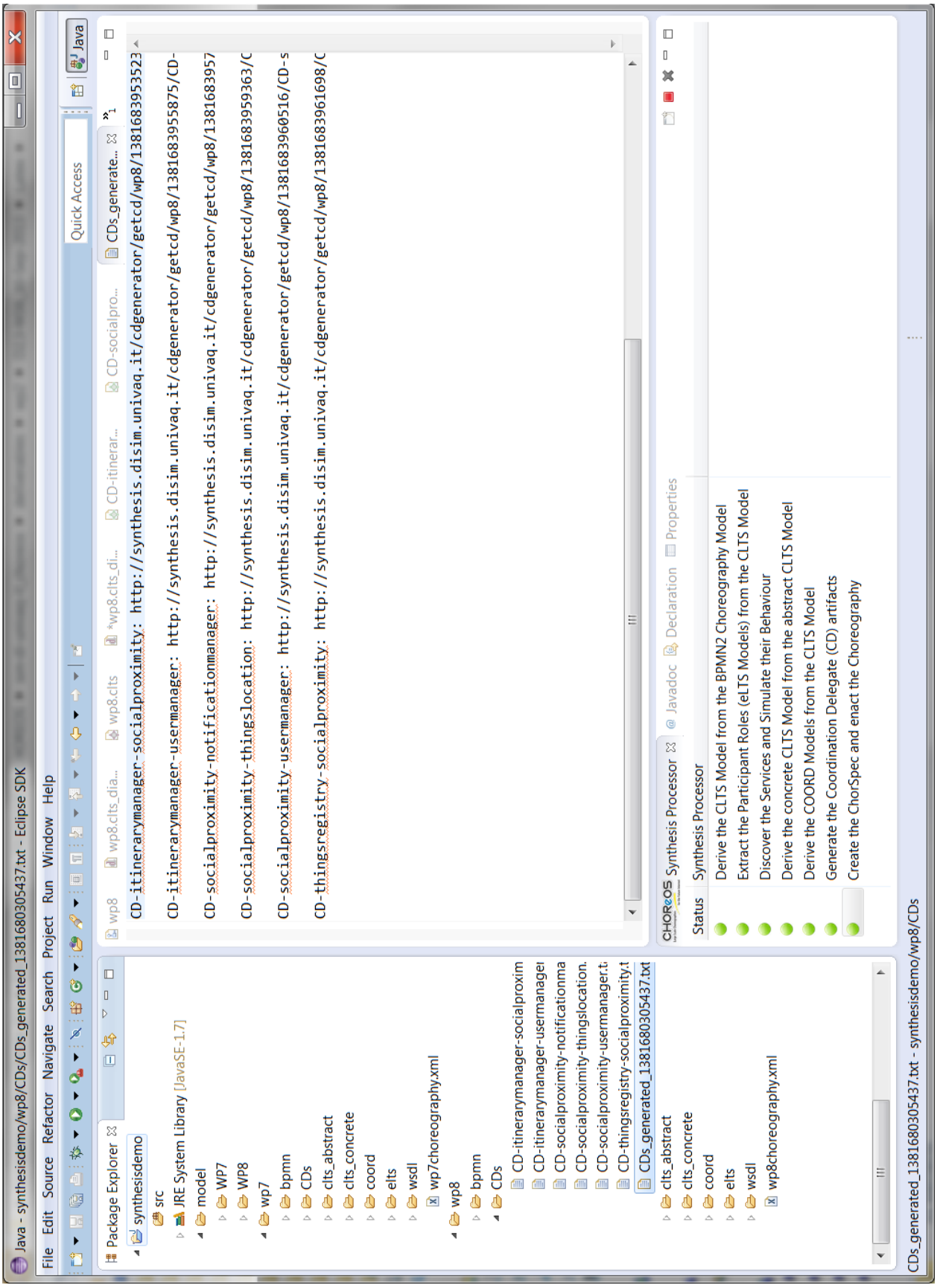


Figure A.42: See the caption of the corresponding figure in the appendix Section A.1