DIP
Data, Information and Process Integration with Semantic Web Services
FP6 – 507483

Deliverable

WP3: Service Ontologies and Service Description
D3.5
An Ontology for Web Service Choreography

Stefania Galizia (OU)
Michael Stollberg (UIBK)
Edward Kilgarriff (NUIG)
Laurent Henocque (ILOG)

January 20, 2006
SUMMARY

This deliverable specifies the ontology for describing Web service choreography interfaces in DIP.

Choreography is concerned with the interactions of Web services with their users in order to consume a Web service. Our approach considers the interface of a Web Service as the most relevant aspect. We define a conceptual model along with a multi-layered language for describing choreography interfaces and related aspects. Following the methodology for defining description models of OMG’s Meta Object facility that is also used in WSMO, this represents a meta-model layer ontology.

The deliverable contains:

- requirements analysis for choreography-related descriptions in Semantic Web services
- the conceptual model taken for choreography-related descriptions in DIP
- detailed definitions of the aspects of choreography-related descriptions in Semantic Web services and respective modeling notions defined within DIP
- an extensive example for modeling choreography-related descriptions within the commonly referred shipper-producer usage scenario

The deliverable is accompanied by the document "DIP Interface Description Ontology" that specifies the common description language for choreography and orchestration interfaces.

Serving as the specification for describing choreography-related aspects of Semantic Web services within the DIP project, this document is expected to be used in the following DIP deliverables:

- D4.11 WSMO Studio v2
- D4.15 Goal-oriented composition prototype
- D5.3a Business process level mediation module specification
- D5.4 Business data and process-level mediation module prototype v1
- D5.5 Business data and process-level mediation module prototype v2
- D6.13 Architecture prototype v3.0
- D6.14 WMSO API v2.0
- D9.10 GIS WSMO descriptions
- D9.11 GIS prototype v 1.0
- D9.12 GIS prototype v 2.0
Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
**Document Information**

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>Acronym</th>
<th>Full Title</th>
<th>Project URL</th>
<th>Document URL</th>
<th>EU Project Officer</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP6 – 507483</td>
<td>DIP</td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td><a href="http://dip.semanticweb.org/">http://dip.semanticweb.org/</a></td>
<td></td>
<td>Kai Tullius</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
<th>Work Package Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>An Ontology for Web Service Choreography</td>
<td>3</td>
<td>Service Ontologies and Service Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M24</td>
<td>31-Dec-05</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Nature</th>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>version 0.5</td>
<td>prototype</td>
<td>public</td>
</tr>
<tr>
<td>final</td>
<td>report</td>
<td>consortium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>E-mail</th>
<th>Partner</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stefania Galizia</td>
<td><a href="mailto:S.Galizia@open.ac.uk">S.Galizia@open.ac.uk</a></td>
<td>Open University</td>
<td>+44 (0) 1908-659461</td>
</tr>
</tbody>
</table>

**Abstract (for dissemination)**

The objective of work-package 3 will be to employ the ontology and Semantic Web infrastructure by adding semantics to the web service description. This deliverable D3.4 addresses orchestration and business process ontology that is an important component for the Semantic Web and Web Services usage and as consequence, used for several purposes in DIP. In this context, the D3.4 aims to define the formalisms and build an orchestration and business process ontology that will provide the basis for advanced querying, reasoning and constraints implementation to be used in web services composition.

**Keywords**

Choreography, Web Service Interface, Description
## Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-10-05</td>
<td>1</td>
<td>Michael Stollberg</td>
<td>document creation and first editing</td>
</tr>
<tr>
<td>31-10-05</td>
<td>2</td>
<td>Stefania Galizia</td>
<td>outlined Choreography Description Ontology section</td>
</tr>
<tr>
<td>01-11-05</td>
<td>3</td>
<td>Michael Stollberg</td>
<td>finished first draft editing</td>
</tr>
<tr>
<td>31-10-05</td>
<td>4</td>
<td>Stefania Galizia</td>
<td>added future work section and moved translation uml to asm in that section</td>
</tr>
<tr>
<td>09-11-05</td>
<td>5</td>
<td>Mathias Kleiner</td>
<td>added UML2AD choreography metamodel and example figures for UML2AD</td>
</tr>
<tr>
<td>10-11-05</td>
<td>6</td>
<td>Laurent Henocque</td>
<td>added support for Z specification and first step in specifying the Workflow Metamodel</td>
</tr>
<tr>
<td>10-11-05</td>
<td>7</td>
<td>Michael Stollberg</td>
<td>changed structure wrt new DIO common document</td>
</tr>
<tr>
<td>10-11-05</td>
<td>8</td>
<td>Laurent Henocque</td>
<td>first complete UML2AD metamodel specification</td>
</tr>
<tr>
<td>17-11-05</td>
<td>9</td>
<td>Stefania Galizia</td>
<td>added Choreography in IRS-III in the section concerning tools</td>
</tr>
<tr>
<td>27-11-05</td>
<td>10</td>
<td>Stefania Galizia</td>
<td>small changes in the section concerning tools</td>
</tr>
<tr>
<td>28-11-05</td>
<td>11</td>
<td>Michael Stollberg</td>
<td>further writing</td>
</tr>
<tr>
<td>28-11-05</td>
<td>12</td>
<td>Stefania Galizia</td>
<td>“Summary”, not more “Executive Summary”; added Disclaimer; removed IRS architecture</td>
</tr>
<tr>
<td>13-12-05</td>
<td>13</td>
<td>Stefania Galizia</td>
<td>Most corrections suggested by James</td>
</tr>
<tr>
<td>28-12-05</td>
<td>14</td>
<td>Michael Stollberg</td>
<td>Corrections and editing accord. to reviewers’ comments</td>
</tr>
<tr>
<td>16-01-06</td>
<td>15</td>
<td>Stefania Galizia</td>
<td>Corrected some typos</td>
</tr>
<tr>
<td>19-01-06</td>
<td>16</td>
<td>Stefania Galizia</td>
<td>Updated template</td>
</tr>
</tbody>
</table>

## Reviewers

<table>
<thead>
<tr>
<th></th>
<th>E-mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marin Dimitrov</td>
<td><a href="mailto:marin.dimitrov@ontotext.com">marin.dimitrov@ontotext.com</a></td>
<td>+359(2) 976-8308</td>
</tr>
<tr>
<td>Partner</td>
<td>Sirma</td>
<td>+43 (512) 507-6450</td>
</tr>
<tr>
<td>James Scicluna</td>
<td><a href="mailto:james.scicluna@deri.org">james.scicluna@deri.org</a></td>
<td>+43 (512) 507-6450</td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway | NUIG | Dr. Sigurd Harand  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: sigurd.harand@deri.org  
Tel: +353 91 495112 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32,  
10117 Berlin, Germany  
E-mail: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc. | BT | Dr. John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
E-mail: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
E-mail: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
<table>
<thead>
<tr>
<th>Institution</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA                                         | Christian de Sainte Marie  
9 Rue de Verdun, 94253,  
Gentilly, France  
E-mail: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG                                       | Torsten Schmale, inubit AG,  
Lützowstraße 105-106  
D-10785 Berlin,  
Germany  
E-mail: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A.          | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
E-mail: rbenjamins@isoco.com  
Tel: +34 913 349 797 |
| NIWA WEB Solutions                             | Alexander Wahler  
NIWA WEB Solutions  
Niederacher & Wahler OEG,  
Kirchengasse 13/1a  
A-1070 Wien  
E-mail: wahler@niwa.at  
Tel.: +43 131 95843 11 |
| The Open University                             | Dr. John Domingue  
Knowledge Media Institute,  
The Open University, Walton Hall,  
Milton Keynes, MK7 6AA, UK  
E-mail: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG                                          | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
E-mail: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma AI Ltd.                                   | Atanas Kiryakov,  
Ontotext Lab, - Sirma AI EAD,  
Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse,  
Sofia 1784, Bulgaria  
E-mail: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
| Unicorn Solution Ltd. | Unicorn | Jeff Eisenberg  
| Unicor Solutions Ltd,  
| Malcha Technology Park 1  
| Jerusalem 96951,  
| Israel  
| E-mail: Jeff.Eisenberg@unicorn.com  
| Tel.: +972 2 6491111 |
| Vrije Universiteit Brussel | VUB | Pieter De Leenheer,  
| Starlab- VUB  
| Vrije Universiteit Brussel  
| Pleinlaan 2, G-10  
| 1050 Brussel, Belgium  
| E-mail: Pieter.De.Leenheer@vub.ac.be  
| Tel.: +32 (0) 2 629 3749 |
# Table of Contents

1 **Introduction**  

2 **Requirements and Aims**  
   2.1 The Big Picture  
   2.2 Requirements for Choreography Interface Descriptions  
   2.3 Approach for Choreography Interface Descriptions  

3 **Choreography Interface Descriptions**  
   3.1 Choreography Interface Properties  
      3.1.1 External Visible Behavior for Service Consumption  
      3.1.2 Communication Structure  
      3.1.3 Grounding  
   3.2 Client and Service Choreography  
   3.3 Intended Tooling Support  
      3.3.1 DIP Tools for Choreography  
      3.3.2 WSMX Choreography Execution Engine  
      3.3.3 The Internet Reasoning Service IRS  

4 **Illustrative Example**  
   4.1 The Shipper-Producer Usage Scenario  
   4.2 Choreography Interface Descriptions with UML2 Activity Diagrams  
   4.3 Choreography Interface Descriptions with Ontologized ASMs  

5 **Conclusions**  

Deliverable 3.5  

January 20, 2006
1 INTRODUCTION

This deliverable specifies the description ontology for choreography interfaces and related aspects of Semantic Web services for the DIP project.

Choreography is concerned with the interactions of Web services with their users in order to consume a Web service. It is our understanding that the most relevant aspect therein is the interface of a Web service that describes how a client needs to interact with the Web service in order to consume its functionality. We refer to this as the **Choreography Interface** of a Web service. This document introduces and explains the approach for defining choreography interfaces along with the specification of the description language in DIP. Following the methodology for defining description models for multi-layered software architectures defined in OMG’s Meta Object facility [17] that is also used for specifying WSMO, this represents a meta-model layer ontology.

The approach presented here follows requirements from the conceptual perspective for comprehensive meta-model description ontologies for Semantic Web services and from user requests, and takes technical requirements for related DIP technologies into account. While we discuss this in detail in the subsequent sections, the requirements can be summarized as follows: (a) support for ontologies as the underlying data model for the information interchanged, (b) appropriate representation of the communication process excepted and supported by a Web service for consuming its functionality, (c) an appropriate formal model with clearly defined semantics in order to allow reasoning on choreography related descriptions of Semantic Web services, (d) grounding to Web service technologies in order to allow invocation and consumption of Web services, and (e) a suitable graphical representation of choreography related descriptions of Semantic Web services.

On basis of this, we define the conceptual model for choreography along with description languages as follows. The choreography interface of a Web service defines the external behavior of the service for consuming its functionality. This encompasses those aspects of the internal business process of the Web service wherefore interaction with the client is needed for consuming the service functionality. It is described by the process expected, respectively supported for information interchange, and by the content of messages interchanged which is defined on basis of ontologies. In correspondence to the choreography interface of a Web service, a service requester (client) defines a so-called **Client Choreography**. While the former describes all possible paths of interaction with a Web service, the latter describes one path of interaction that allows the client to achieve his objective by consuming a Web service via its choreography interface. Therefore, the client choreography needs to be compatible with the choreography interface of the Web service that is to be used.

This deliverable is accompanied by the document “DIP Interface Description Ontology” that specifies the common description language for choreography and orchestration interfaces. This layered language consists of a low-level formal model, namely so-called ‘ontologized Abstract State Machines’ as the description model for Web service interfaces defined within the Web Service Modeling Ontology WSMO [27]. Abstract State Machines (ASMs) are used as the underlying formalism for specifying the dynamics of choreography interfaces, which provide a rich and flexible formalism with minimal ontological commitment. WSMO ontologies are inherently supported as the data model for information that are interchanged. For graphical representation, UML2 Activity Diagrams are used whose semantics are defined by automatically translating
them to ontologized ASMs. The intended usage scenario is that a service provider or requester models the choreography interface as UML2 Activity Diagrams, supported by a graphical tool. Then, this definition is translated into the formal model. In consequence, both the UML2 as well as the ASM representation can be used as interchange formats for choreography related descriptions in DIP.

The work presents an extension of the respective work within the Web Service Modeling Ontology WSMO. While all aspects concerning the description language are located in the accompanying document “DIP Interface Description Ontology”, this deliverable is concerned with the conceptual aspects of choreography related descriptions and their usage within Semantic Web services. We also discuss and position our meta-model ontology for choreography within related work. As discussed in the accompanying document in detail, exiting current Web service technologies for choreography have deficiencies in several aspects that we consider to be crucial. Hence, our model presents a novel approach for semantically describing choreographies and related aspects in Web services.

The document is structured as follows: Section 2 reveals the requirements for choreography related descriptions of Web services and explains the approach taken for the DIP project; Section 3 discusses in detail the relevant aspects for choreography interface descriptions and the intended tool support to be developed in the DIP project; Section 4 provides a comprehensive example for modeling choreography descriptions of Semantic Web services, and Section 5 concludes the deliverable. The accompanying document “DIP Interface Description Ontology” provides the specification of the description language for choreography interfaces.
2 Requirements and Aims

This section introduces the approach taken for specifying the meta-model layer ontology for choreography related descriptions in DIP. We therefore first allocate this within the overall description framework for Semantic Web services, then gather the requirements for choreography interface descriptions, and finally outline the approach taken within DIP.

2.1 The Big Picture

Semantic descriptions of Web services shall provide the basis for respective mechanisms for automating the Web service usage process [30]. For semantically describing Web services, respective overall models commonly differentiate between functional descriptions that describe what the service does and behavioral descriptions that define how the service works. Within OWL-S [21], the Service Profile provides the functional description and the Service Model in conjunction with the Service Grounding provides the behavioral description. The Web Service Modeling Ontology WSMO [25] defines the capability of a Web service as its functional description and so-called interfaces as the behavioral description. WSMO being chosen as the basis for DIP, we recall WSMO Web service description in order to identify the characteristics and to determine requirements for choreography related descriptions in DIP.

WSMO defines a description model that is intended to encompass the information needed for automatically determining the usability of a Web service. As shown in Figure 2.1, a WSMO Web service description is comprised of four elements: (1) non-functional properties, (2) a capability as the functional description of the service; summarized as service interfaces, (3) a choreography that describes the interface for service consumption by a client, and (4) an orchestration that describes how the functionality of the service is achieved by aggregating other Web services. These notions describe the functionality and behavior of a Web service, while its internal implementation is not of interest.

The relevant aspects in this context are the notions of choreography and orchestration. These are concerned with the behavior of Web services for consuming their functionality, respectively for achieving the functionality of a Web service by using and aggregating other Web services. The W3C defines choreography and orchestration as follows [19]:

**Choreography** = Web Services Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, may be other Web Services, applications or human beings.

**Orchestration** = An orchestration defines the sequence and conditions in which one Web Service invokes other Web Services in order to realize some useful function. That is, an orchestration is the pattern of interactions that a Web Service agent must follow in order to achieve its goal.

Refining these general definitions, we understand the respective interfaces of a Web service as the most important aspects. Such an interface describes how a Web
A service can interact with its environment with respect to the process of information interchange expected for consumption of the service functionality, respectively the process of information interchange and coordination of other Web services used for achieving its functionality [4]. The rationale for this conceptual model is that Web service usage and interaction is happening in a peer-2-peer manner, meaning that information interchange and cooperation is controlled by the Web services as the peers without any central control unit. A main implication of our conceptual model is that we do not need to describe client-service, respectively service-service interaction from a global perspective.

In consequence, we follow the conceptual model of WSMO [25] by defining two types of interfaces for describing Web services. The so-called Choreography Interface that describes the behavioral interface of a Web service for interaction with a client that wants to consume the Web service functionality, and the Orchestration that defines the behavioral interface of a Web service for achieving its functionality by using and aggregating other Web services. Figure 2.2 shows the overall picture of these notions and their intended usage within Semantic Web service technology.

In a generic scenario, a Client wants to use a Web service \( WS \) for achieving some objective. In order to consume the functionality of the Web service, the client interacts with \( WS \) in accordance to its choreography interface. This describes the information interchange expected and supported by \( WS \) with respect to the process and content of the information sent and received by the Web service. A choreography interface is mandatory for a Web service description in order to allow consumption of its functionality. Moreover, \( WS \) might use other Web services \( WS_2 \) and \( WS_3 \) in order to achieve its functionality, meaning that not the complete functionality is realized within the implementation of \( WS \) but obtained by utilizing functionalities provided by other Web services - which in fact is a major promise of Web service technology [1]. The way how \( WS \) interacts with \( WS_2 \) and \( WS_3 \) is described in the orchestration of \( WS \). Thereby, \( WS \) consumes \( WS_2 \), respectively \( WS_3 \) via their choreography interfaces; hence, from the perspective of \( WS_2 \), \( WS \) appears as a client in the same way as an end-user.

It is important to remark that the choreography and orchestration of a Web service
represent interfaces. Instead of describing and handling client-service as well as service-service interaction from a global perspective, we envision advanced conformance checks for determining the feasibility of interaction execution \[29\]. This deliverable is concerned with choreography interfaces, while the description model for orchestrations is defined in deliverable D3.4 “An Orchestration and Business Process Ontology”. Before examining the properties and requirements for suitable semantic descriptions of choreography interfaces below, we summarize the our conceptual model.

- information interchange for consumption and cooperation of Web services happens in a peer-2-peer manner wherein Web services denote the peers that control the interaction instead of a central control unit

- hence, in correspondence to WSMO, we define two types of Web service interfaces: the *choreography interface* that describes the interaction behavior of a Web service for consuming its functionality, and the *orchestration* that describes how a Web service aggregates other Web services in order to achieve its functionality

- a choreography interface description is mandatory for a Web service description in order to allow service consumption by clients; an orchestration is optional in dependence on how the service provider decides to realize the functionality of a Web service

- the compatibility of the behavioral interfaces of clients and services that are supposed to interact is determined by respective conformance tests; this requires a sound formal model for choreography and orchestration interface descriptions

- all client-service interaction for consuming a Web Service happens via the choreography interface of the service

- when a Web Service \(A\) uses another Web Service \(B\) in its Orchestration, \(A\) consumes \(B\) via the choreography interface of \(B\); here, \(A\) represents a ‘normal’ client to \(B\), meaning that \(B\) is not aware of being used in an orchestration of another Web Service (and does not need to be).

---

**Figure 2.2: Choreography and Orchestration Interfaces of a Web Service**
2.2 Requirements for Choreography Interface Descriptions

We determine the following requirements of describing choreography related aspects for Semantic Web service. These result from the overall description model developed within DIP for semantically describing Web services, from end-user requirements gathered, as well as from development plans for execution and reasoning technologies dealing with choreography.

1. **Behavior Interface for Service Consumption**
   In accordance to the general model of Web services recalled above, the choreography interface of a Web service defines the interaction behavior of the service for consuming its functionality. This denotes the public process of a Web service, i.e. the external behavior that is visible for clients in order for automated consumption [6]. The description only covers those aspects where interaction with the client is necessary in order to consume the service functionality. Such interfaces are commonly referred to as behavior interfaces [4].

2. **Mandatory Description Element of Web Services**
   Following the initial idea of Web services [1], they shall provide seamless access interfaces to computational facilities accessible on the Web. This is achieved by standardized descriptions of behavioral interfaces of Web services, wherefore the initial description technique recommended by the W3C is the Web Service Description Language WSDL [8]. In order to provide the basis for ad-hoc usage and combination as the ultimate aim of Semantic Web services, we consider choreography descriptions to be mandatory.

3. **Communication Structure**
   In order to provide the basis for ad-hoc service usage and automated invocation, knowledge about the information to be interchanged as well as their order is required [6]. While the former refers to messages or - more general - communicative acts between interaction partners, the latter is concerned with the communication process that is supported or intended by the Web service or a requester. Both aspects should be described in an integrated manner.

4. **Ontologies as Data Model**
   With respect to the overall design of Semantic Web services, ontologies are to be used as the underlying data model (see [14], [22]). This means that all resource descriptions are to be based on ontologies, and all data elements interchanged between clients and Web services are to be ontology instances. This inherently ensures support for the Semantic Web and provides the basis for semantic interoperability as well as advanced information processing [12].

5. **Executable Web Service Communication Technology**
   In order to enable execution of Web service consumption by clients, choreography interface descriptions need to be aligned with respective Web service technologies. Commonly referred to as grounding, it is desirable not to restrict this to
certain technologies; in order to allow usage of existing Web services, standards like WSDL shall be supported.

6. **Graphical Representation**
   In order to provide sophisticated support for definition of choreography interfaces and related aspects, a suitable graphical representation should be supported. Editing, browsing, and maintenance of interface definitions in this graphical user language should be supported by respective Web service management tools.

7. **Formal Description**
   The ultimate aim of Semantic Web services is to enable advanced, inference-based mechanisms for automating the Web service usage process. Regarding both choreography and orchestration within our approach, one central reasoning task commonly is to determine whether for service consumption the interaction of a client and a Web service can be achieved successfully, respectively the interaction of Web services for achieving a functionality via an orchestration [29]. Another reasoning task is concerned with resolving process level mismatches that hamper successful interaction [9].

   Such techniques represent the actual benefit obtainable by semantic descriptions of Web services which are not supported by existing syntactic service descriptions. The pre-requisite therefore is a sound formal model with unambiguous semantics for describing the dynamics within choreography interfaces and orchestrations. This formal model should be appropriate with respect to the process structures that occur in Web service interface descriptions, and it should integrate ontologies as the data model for the information that are to be interchanged.

8. **Higher-level Process Constructs**
   In order to allow definition of more complex dynamic constructs of the communication structure in choreography interface descriptions and in order to support advanced handling of these, higher-level process constructs should be supported. In particular, general workflow patterns as defined in [32] should be taken into consideration.

2.3 **Approach for Choreography Interface Descriptions**

   With respect to the above examinations, we define a choreography interface as follows:

   *A choreography interface is a mandatory description element that specifies the behavior of a Web service or a Web service request for service consumption by describing the external visible communication structure (i.e. communicative acts to be interchanged and the process of this) along with a grounding towards Web service communication technology.*
While we discuss the aspects of choreography interfaces and their description in the next section in more detail, the following depicts the structure of the layered description languages that we develop with respect to the requirements determined above. This consists of the following elements whose interrelations are depicted in Figure 2.3.

Starting from the top, we have chosen UML2 Activity Diagrams as the graphical user language that is to be supported by respective tools for editing and managing Web service interface descriptions. Ontologies are used in the UML2 Activity Diagrams as the data model for the information that is to be sent or received within an interface definition. Apart from being a suitable graphical user language, UML2 Activity Diagrams support higher-level process constructs as required.

As the formal model for semantically describing choreography interfaces, we use so-called ontologized Abstract State Machines as the approach developed within WSMO for formally describing the dynamic semantics of Web service interface definitions [27]. While explaining the structure and semantics of this language in the accompanying document “DIP Interface Description Ontology” in detail, we denote the following benefits of this approach: (1) ontologies are inherently supported as the data model, (2) the approach is inherently integrated with other aspects for semantically describing Web services as defined in DIP, and (3) the language serves as a basis for execution and reasoning technologies for client-service as well as service-service interaction as planned in the DIP project.

It is to remark that this is the common language for describing behavior interfaces of Semantic Web services in DIP. This means that the same language constructs are used for describing choreography interfaces as well as orchestrations as specified in the accompanying language specification document “DIP Interface Description Ontology” in detail.
3 Choreography Interface Descriptions

On basis of the preceding examinations, this section explains conceptual aspects of choreography interface descriptions in DIP. We first discuss the relevant aspects of choreography interface descriptions, then introduce the conceptual distinction of client and service choreography interfaces with respect to goal-driven Web service usage and how these choreography descriptions can be created, and finally outline the intended tooling support for choreography descriptions in DIP.

3.1 Choreography Interface Properties

Above, we have introduced the definition of choreography interfaces in DIP to be mandatory behavioral interface of a Web service or a service request for consuming a Web service by describing the external visible communication structure (i.e. communicative acts to be interchanged and the process of this) along with a grounding towards an executable Web service communication technology. This shall meet all the requirements identified in Section 2.2 as follows:

- the conceptual requirements (1 - Behavior Interface), (2 - Mandatory Description Element), (3 - Communication Structure), and (5 - Grounding) are inherently incorporated in the definition
- the requirements (4 - Ontologies as Data Model), (7 - Formal Model), (8 - Higher-level Process Constructs) and (6 - Graphical Representation) refer to the description language that is defined in the accompanying document “DIP Interface Description Ontology”.

In order to clarify and rationalize this definition we need to examine the following three conceptual features in model detail: the External Visible Behavior as those aspects of the workflow of a Web Service or a client where interaction with counterpart is required, the Communication Structure as the order and process of communicative acts that are to be interchanged, and the Grounding towards a communication technology for interaction and information interchange between Web services over the Internet. The following addresses each one in detail.

3.1.1 External Visible Behavior for Service Consumption

This aspect is concerned with what needs to be specified in a choreography interface. In accordance to the underlying model of our approach, the choreography interface of a Web service or a service request describes the entity’s individual behavior for participating in an information interchange for consuming a Web service. This relates to those aspects of the workflow of a Web Service where interaction with the client is required so that the client can consume the Web Service functionality.

In order to clarify this let’s consider the well-known example of a Virtual Travel Agency, see [31] for a detailed use case description. The general setting is that a Web service VTA provides a general purpose travel service for booking hotels, flights, train- and other transportation tickets. This VTA Web service dynamically uses and combines other Web services in order to resolve specific requests.
Among the utilized Web services, there is a Web service of the national Austrian railway operator ÖBB that offers search and booking facilities for train tickets in Austria. Its business process is as follows. The ÖBB Web service receives a request for some trip, and returns a set of possible connections as offers for purchase (search facility); if a request for purchase is received, the service asks for payment information; the service terminates by providing a purchase contract for the desired trip (booking facility). Figure 3.1 shows the structure of the choreography interface of the ÖBB Web service. According to its business process, this consists for four sequential activities invocation, connection choice, purchase contract, and payment and delivery. Each activity correlates to one or more messages that are expected or supported to be interchanged, denoted on the left hand side of the figure; the right hand side of the figure shows the conditions under which a subsequent activity is reached in the right hand side of the figure.

![Figure 3.1: Choreography Interface Structure Example](image)

While discussing communication related aspects below in more detail, it is important to note which information is specified in the choreography interface. Let’s therefore study the first activity invocation in detail. The service consumption is initiated when it receives a message with a buyer information and an itinerary (i.e. a desired trip with start and end station, travel date, etc.). Then, the Web service implementation checks validity of the input and determines possible itineraries for the requested trip; the technical realization - i.e. whether the functionality is implemented as program or achieved by using other Web services - is not of interest at this point. The result of the internal technical realization is then communicated to the client by an outgoing message in the connection choice activity.

Now, what we model in the choreography interface as the external visible behavior of the Web service is the messages send and received and the conditions under which
transitions to subsequent activities occur. This means, we specify that the connection choice activity will be processed next if the input is valid, otherwise an error message will be created and the service usage is stopped. We do not specify internal, private decision or computations of the Web service, e.g. how possible itineraries are determined.

Within the world of business process technologies, this is commonly referred to as the public process of an entity [6]. Within the world of Web service technologies, the Web service description element that specifies its external visible behavior for consumption by a client are referred to as behavior interfaces [4]. As outlined in the requirements above (see Section 2.2), such interfaces need to specify the communication behavior and are considered as a mandatory description element for Semantic Web services in order to allow automated service invocation and usage [30].

3.1.2 Communication Structure

The second aspect to be examined in more detail is the communication structure defined in a choreography interface. This is comprised of messages as single communicative acts interchanged between Web services and clients, and of their order as the process of when and under which conditions which message exchange is expected or provided by a choreography interface. We discuss both aspects and explain how our approach addresses the respective requirements.

Messages

Regarding messages, the Shannon and Weaver’s theory of communication that serves as a basis for design of communication technologies distinguishes three levels of information [28]. The first one is the level of content that is concerned with the information interchanged, i.e. the content of a message (e.g. information on the buyer in the first message in Figure 3.1). Here, information is understood to be data that has a concrete meaning in a context, i.e. data that has a semantic definition in a domain of discourse. In order to ensure this within communication and information interchange of Web services, our model applies ontologies as the data model for the content level.

The second level concerns the relationship of the communication partners, i.e. the communicative roles of entities participating in an interaction. Within multi-entity communication, we need to clearly identify the sender and receiver of a message as well as the nature of the communication. The latter refer to synchronous or asynchronous patterns of communication [26]. Synchronous communication means that the progression of both entities depends on the information interchange, i.e. none of the partners can continue its individual process without completing the information interchange; asynchronous communication denotes the opposite type communication (i.e. a notification is broadcasted that does - at least not directly - influence continuous processing of entities). Our approach for describing choreography interfaces addresses this level as follows:

- a choreography interface supports binary interaction, i.e. between one Web service and one client (several of these binary interactions might take place concurrently). Hence, the identification of sender and receiver of a communicative act can be represented by denoting the direction of the message (in- or outgoing)
• the characteristic of a message with respect to synchronous and asynchronous communication can be represented in the conditions of state transitions within Abstract State Machines (see accompanying document “DIP Interface Description Ontology” for details).

The third level is concerned with the communicative intention of a message. This refers to what the sender of a message expects the receiver to do with it. Therefore, collections of communicative acts have been identified, e.g. request for requesting some information or tasks from the partners, response as the reply to a request, inform for notification, etc. These are mainly applied within Agent Communication Languages (short: ACL) like KQML or FIPA ACL in order to handle cooperative communication and information interchange between intelligent, autonomously acting software agents. This information enables intelligent software agents to determine their interaction behavior autonomously with respect to changes in the environment. In contrast, within Web services the intention level of communication appears to be dispensable because the information interchange is only needed for executing a Web service as a passive software component. In consequence, all messages that are expected or provided by a Web service within one concrete path of interaction are required and necessary in order to complete service consumption correctly. However, additional information on this description level can be applied for managing the interactions between clients and Web services as we explain in section 3.3.3 in more detail.

The Process of Communication

Apart from the messages expected or provided by a Web service, a choreography interface needs to specify when each message is to be communicated - i.e. the process of communication expected by the Web service in order to consume its functionality. While we have determined above that this process is the external visible behavior or public process of a Web service, we now investigate how to present this as a process description.

The major deficiency of the Web Service Description Language WSDL - which is the W3C recommendation and quasi-standard for Web service invocation and usage at this point in time - is support for this aspect. The WSDL description of a Web service consists of a collection of operations that define one-shot interaction for consuming a Web service - but no information can be specified on the order or the process of which operation to be invoked at which state of consuming the Web service. While the current W3C efforts around choreography develops the Web Service Choreography Description Language WS-CDL for describing multi-party interactions from the global perspective, the first but uncompleted W3C effort for adding the desirable process information to WSDL descriptions was the Web Service Choreography Interface (WSCI) along with formal semantic on basis of process algebra defined in 5. Also, other efforts like abstract processes within BPEL4WS are concerned to behavior interface descriptions of Web services.

1This opposes our approach: while we consider the behavior interface of a Web service for participating in an interaction as the most important aspect, the WS-CDL perspective only considers the global business process for multi-party interactions. See the related work section in accompanying document “DIP Interface Description Ontology” for a detailed discussion.
However, all these approaches have deficiencies in one or another aspect that we have determined to be important - for instance not supporting ontologies as the data model or lacking in clear semantics for process descriptions. While we discuss and position our approach within related work in detail in the accompanying document “DIP Interface Description Ontology”, we here summarize the relevant properties of the Process of Communication in Web service interface descriptions and how this is addressed within our approach.

The external visible process for consuming a Web service as described within a choreography interface has one or more initial states where the interaction for service consumption can commence from. Then, an arbitrary sequence of states or actions is executed until a termination state is reached [20]. Being the public process of the Web service for its consumption, the progression of this process (i.e. the transitions between states) is determined by the information received and sent by the Web service. In particular, the message content determines the path followed during an interaction as the information received or produced by the Web service determine possible state transitions with respect to the transition conditions specified. So we can say that the progression of the communication process in a choreography interface is determined by the ontology instances interchanged during service consumption. These observations have determined the development of ontologized Abstract State Machines as the formal model for describing Web service interfaces within WSMO [27].

3.1.3 Grounding

The final aspect of choreography interfaces to be investigated in detail is executable communication and information exchange technologies for Web services. Naturally, this shall take place over the Internet. Currently, DIP interface descriptions are grounded to WSDL [8], as specified in detail in the accompanying document “DIP Interface Description Ontology”. WSDL is a W3C working effort that is widely adopted, including tool support for automated generation of WSDL descriptions from program code of Web service implementations (e.g. Apache AXIS, see http://ws.apache.org/axis/). It realizes message-based communication via SOAP [15] for exchange of XML data over the Web. The grounding realization for DIP interface descriptions is specified in the accompanying document “DIP Interface Description Ontology” in detail.

A main benefit of our approach is its modular design that allows replacing the target technology of the grounding by simply changing the grounding specification without need of changing the interface description. As a possible future Web service communication technology, so-called Triple Space Computing is envisioned that aims at overcoming the deficiencies of message-based communication technology for Web services [7], [13]. Extending the concept of tuple space computing - wherein communicating parties write information to be exchanged into a common information space wherefrom the respective participants read the data - with ontologies as the data model, Triple-Space Computing co-aligns with the Web design paradigm of persistent publishing and reading of information that enable Web scale information interchange.
3.2 Client and Service Choreography

In order to support goal-driven Web service usage as one of the objectives of WSMO and DIP [30], we distinguish two types of choreography interface descriptions, namely *Client Choreography Interfaces* and *Service Choreography Interfaces*. While the latter refers to the behavioral interface of a Web service for consuming its functionality as defined above, the former denotes the behavioral interface of a client for consuming a Web Service in order to achieve his objective. Figure 3.2 shows both choreography interface types with further explanations in the following.

![Figure 3.2: Client and Service Choreography Interfaces](image)

A Service Choreography Interface describes all possible paths of interaction with the Web service in order to consume its functionality. A path denotes the sequential order of states that are reached during a concrete service consumption session. The right-hand side of the figure illustrates a Web service choreography interface with two possible start states and two possible termination states. Imagine this to be the choreography interface of some search and sell service like Amazon: the left start state denotes a start state for buying a product, and the right one the start state for searching for a product. While the former one will always result in a termination state where a product is purchased (the colored termination state), the latter start state might result in purchasing or in providing a product search result (the not colored termination state). In order to be a correct description of a Web service’s behavior for consuming its functionality, its service choreography interface needs to define all possible paths of consumption.

In contrast, a Client Choreography Interface describes one path of interaction for consuming a Web service via its choreography interface that results in a state wherein the client objective is achieved. In the left hand side of the figure, there is one sequential path of interaction denoted in the client choreography interface that corresponds to the colored path in the choreography interface of the Web service. Imagine the client objective to be that of buying a certain product, the client choreography interface describes the structure of the communication behavior that the client can perform in order to obtain the desired functionality from the Web service. Thereby, the client choreography interface needs to be compatible with at least one possible path of interaction within the choreography interface of the Web service to be consumed.

The reason for this differentiation is that a client should not be requested to provide a complete description of the desired interaction behavior. Besides, this would hamper consumption of possibly usable Web services because of unresolvable mismatches in the choreography interface of a client and a Web service. As a more appropriate
conceptual model, it is expected that the client choreography interface as defined above is determined with respect to (1) the choreography interface of the Web service that is to be consumed, and (2) the input as well as additional constraints that the client defines for Web service consumption. We understand this as an appropriate extension of the WSMO conceptual model. Note that the description language is the same for both client as well as service choreography interfaces.

Regarding the creation of service and client choreography interfaces, the former can either (a) be ‘added’ in sense of an annotation of an existing Web service implementation with a WSDL description, or (b) be ‘specified’ in sense of a formal specification of the service behavior for functionality consumption. In contrast, we envision semantically enabled mechanisms for automated generation of client interfaces for a given request (i.e. a goal or another usage scenario) and a Web service that has been determined for application for resolving the request. Respective techniques are under consideration at the time of writing.

3.3 Intended Tooling Support

Concluding the investigations on choreography descriptions and their usage within Semantic Web services, the following outlines the intended support for creating, maintaining, and executing choreography descriptions with the DIP project. We first summarize the respective technology developments and then outline the planned development for choreography execution technology within WSMX and IRS as the DIP technology platforms.

3.3.1 DIP Tools for Choreography

There are two types of tools and technologies planned within DIP that deal with choreography descriptions. Naturally, these will be integrated into the DIP Architecture (DIP deliverable D6.13). These tools are to be tested within the use case within DIP work package 9.

WSMO Studio

WSMO Studio (DIP Deliverable D4.11) develops an integrated end-user tool suite for editing and managing Semantic Web services on basis of the WSMO specification. It is planned to have a plug-in for editing, browsing, and managing choreography and orchestration descriptions following the specification provided in this deliverable and related specification documents.

Tools

Three tools are planned to be developed in DIP that are concerned with choreography and orchestration descriptions.

1. **Execution Engines**: engines for executing interaction on basis choreography interfaces (DIP Deliverable D4.7 and D54. / D5.5) and for orchestrations (D4.20) are planned. As explained below in more detail, these are designed to work on ontologized Abstract State Machines as the formal model of DIP interface descriptions.
2. **Composer**: the goal-driven Web service composition tool developed in (DIP Deliverable D4.12 and D4.15) works on UML2 Activity Diagrams. The translation from this user language to the formal model as specified in the accompanying document “DIP Interface Description Ontology” ensures the integration of this tool within the DIP architecture.

3. **Process Mediator**: for resolving process-level mismatches between choreography interfaces that might hamper successful interaction of entities, the process mediator developed in DIP Deliverables D5.4 and D5.5 works on ontologized Abstract State Machines as the formal model of DIP interface descriptions.

### 3.3.2 WSMX Choreography Execution Engine

The choreography engine’s responsibility is to support the behavioral aspect of the communication between the requester and provider. It works on the ontologized Abstract State Machine format, utilizing the grounding to WSDL as the communication technology for executing information exchange between clients and Web services. The choreography engine takes the choreography descriptions of the involved parties and creates state machine instances for each of them. Interactions between the two parties cause guarded transitions to fire in both of these machine instances, within the range of all possible interactions.

In compliance with the DIP / WSMX architecture, the choreography engine tackles a single aspect of the general problem that is orthogonal to all other aspects. Data- and Process Mediation happens transparently to the choreography engine between applying a co-called *updateset* to one of the ontology ASM instances and starting the next step of the other ontology instance (see accompanying document “DIP Interface Description Ontology” for details on the formal semantics of ontologized ASMs). The link to the Invoker, which carries out the actual invocation, manifests itself through an object representation of the grounding information that accompanies the respective transition rule that triggered the update, ultimately causing the invocation.

### 3.3.3 The Internet Reasoning Service IRS

The Internet Reasoning Service [10] (abbreviated as IRS-III or simply IRS) is a framework and implemented infrastructure for Semantic Web services based on WSMO. The following outlines the realization for Web service consumption via choreography interfaces as specified in this deliverable, referring to [11] for details.

**Choreography model**

We assume that IRS clients are able to formulate their request as a goal instance. This means that we only require choreographies between the IRS and the deployed Web services. In IRS-III choreography execution thus occurs from a client perspective. This means that for carrying out a Web service invocation, the IRS executes a Web service client choreography which sends the appropriate messages to the deployed Web service.

A choreography is described in IRS-III by a grounding declaration and a set of guarded transitions. The grounding specifies the conceptual representation of the operations involved in the invocation of a Web Service and their mapping to the implementation.
level. More specifically, the grounding definitions include *operation-name*, *input-roles-soap-binding*, *output-role-soap-binding*. The guarded transitions are the set of rules, which represent the interaction between IRS-III and the Web Service on behalf of an IRS client. They are applied when executing the choreography. This model is executed at a semantic level when IRS-III receives a request to achieve a goal. In the rest of this section we list the main design principles which motivate our choreography model.

**Design principles**

**Ontology Based.** Ontologies form a central pillar of the semantic web. Founding our choreography descriptions on ontologies means that we can refer to relevant domain dependent concepts or relations within guarded transitions.

**IRS as a Broker.** As mentioned earlier, the IRS acts as a broker for capability based invocation. A client sends a request to achieve a goal and the IRS finds, composes and invokes the appropriate Web services. The choreography to the IRS is thereby fixed. We assume that IRS clients are able to formulate their request as a goal instance. This means that we only require choreographies between the IRS and the deployed Web services. Our choreography descriptions are therefore written from the perspective of IRS as a client of the Web service.

**The Predominance of State.** Following the conceptual model outlined above, any message sent by IRS to a Web service will depend on its current state. This includes a representation of the messages received during the current conversation. Choreography descriptions in IRS are represented as ontologized ASMs, i.e. the formal model for describing DIP interfaces. By representing ASM as rules, the sequence of operations and the message pattern instantiations are generated through the evaluation of conditions. A condition is a generic statement on the current situation, for instance, that an error has occurred. The executive part of the guarded transitions updates the state (see accompanying document “DIP Interface Description Ontology” for details on ontologized ASMs).

**Open.** The major components of IRS-III are semantic Web services represented within the IRS-III framework. This feature enables the main functionalities of the IRS to be redefined to suit specific requirements. Following this the IRS choreography engine is itself a semantic Web service.

**Communication Representation.** We have chosen to classify the communication in IRS choreography according to two dimensions, following the system-client cooperation model proposed in KADS [16], namely the *initiative* and the *direction* of the communicative interaction.

The initiative expresses which actor is responsible for starting the communication - i.e. either IRS or the Web service - while the direction represents the communication route which can be from the system to the client or vice-versa. The reason for preferring this communication model is that in this way we can verify at every state which actor has initiative. Initiative is associated with the actor that has control of the conversation. For example, only actors with initiative are allowed to start a conversation or update data previously sent. A message exchange event is a kind of transfer task, an elementary executed operation by an actor during a conversation. According to Greef and Breuker’s communication representation [16], we consider six kinds of events: obtain, present, provide, receive, obtain-initiative, present-initiative. When the IRS does not have the initiative, receive and provide messages are used.
Conversely, obtain and present events occur when the IRS is in control of the conversation. Obtain-initiative and present-initiative allow the initiative to be transferred. For detailed event descriptions see [15].

When a client - that can also be a Web service - invokes the IRS in order to achieve a goal, the choreography engine runs. We depict a simple invocation goal scenario below, underlining the events involved during choreography execution. Figure 3.3 shows the event sequence for this typical goal driven Web service invocation scenario.

![Figure 3.3: A typical sequence of choreography events occurring during goal based Web service invocation in IRS-III](image)

The client initiates the communication with IRS by requesting that a goal be achieved. Within our model this corresponds to receive and obtain-initiative events as the client delegates initiative to the IRS to invoke the required service. During a second phase, the IRS invokes a Web service which returns a response. In this phase the IRS has the initiative and therefore the occurring events are present and obtain.

**Ability to Suspend Communication.** Expectably, there will be situations where it is necessary to suspend the current dialog and resume it later. For example, either the IRS or the Web service may not have some required data or a Web service may go offline.

**Executable Semantic Descriptions.** The semantic representation of choreography interface descriptions should be executable directly or should be able to be compiled to a runnable representation. Our underlying modelling language OCML [23] is operational. Additionally, extensions within the IRS allow us to attach OCML functions to deployed Web services. This means that within a guarded transition one can refer to external data, for example, to “today’s exchange rate”.

**Formalization.** A formal semantics allows us to reason about the choreography descriptions which is useful if we want to automatically compose Web services. For this reason, we adopt ASMs and our formal model is described in the following section.

**Ease of Usage.** If we want our system to be used widely, it is important that the components are easy to use. For this purpose we have defined a relatively small set of choreography specific primitives.

**Choreography primitives**

In IRS are defined a set of choreography specific primitives which can be used in guarded transitions. These primitives provide an easy to use interface to control a conversation between the IRS and a Web service. Developers are also able to include any relation defined with the imported ontologies within guarded transition specifications.
• **Init-choreography.** Initializes the state of the choreography. This primitive runs before a Web service is invoked by IRS-III. At this step the IRS has the initiative and it is ready to start the communication.

• **Send-message.** Calls a specific operation in the Web service. If no inputs are explicitly given IRS obtains the input values from the original goal invocation.

• **Send-suspend.**Suspends the communication between IRS and the Web service, without stopping the choreography executions. This action will occur, for example, when the IRS lacks some data required by a Web service. Executing this primitive suspends the dialog and stores the current state so that communication can be resumed later.

• **Received-suspend.** The communication is suspended by the Web service, when for some reason it is not able to respond to an invocation. As with send-suspend the choreography execution is put on hold. The Web service is free to resume the dialog when conditions allow.

• **Received-message.** Contains the result of a successful send-message for a specific operation.

• **Received-error.** If the execution of a Web service causes an error to occur then the received-error primitive is used. The parameters of received-error include the error message and the type of error which occurred.

• **End-choreography.** Stops the choreography. No other guarded transitions will be executed.

**Execution**

IRS uses a forward-chaining-rule engine to execute a choreography. This means the rules belonging to a choreography are fired according to the state. One important feature of the execution environment of IRS is that it allows the scope of the choreography to be defined for the set of ontologies involved in the Web Service description. The IRS server carries out inferences at an ontological level. During communication with a Web service the ontological level descriptions need to be mapped to the XML based representations used by the specific Web service invoked. We provide two mechanisms which map a) from the ontological level to XML (lower) and b) from XML to the ontological level (lift).

• **Lift.** Lifts an XML string into an ontological construct, represented in OCML. A generic version of this relation is defined within the IRS ontology. SWS developers are free to overwrite this relation inline with the relationship between the results of Web service calls and the ontologies used. The lift primitive has the following input parameters: class-name, web-service-class, xml-string and produces an instance of class-name as output. The semantic developer can thus customize how XML is parsed according the classes within the underlying ontology and the particular Web services selected. In order to cope with XML based input the lift primitive utilizes an in-built SAX based XML parser.
• **Lower.** Lowers the ontological construct to XML. The input parameters to lower are: instance-name and a class web-service. The output is xml-string. As for the lift the XML generated can be customized according to classes within the ontology and the Web service class. For example, the XML generated for instances of a person class may include a full name for one Web service and only a family name for another.
4 ILLUSTRATIVE EXAMPLE

This section provides an exemplary specification of choreography interface descriptions using the developed description languages. For language syntax and semantics, we refer to the accompanying document “DIP Interface Description Ontology” that specifies the common description language for choreography and orchestration interfaces.

4.1 The Shipper-Producer Usage Scenario

For illustration purpose we apply the shipper-producer example defined in [24]. This use case is also used for illustrating orchestration descriptions within deliverable D3.4.

While briefly summarizing the use case setting,

Below we provide the choreography interface descriptions in the DIP interface description languages, that is in UML2 Activity Diagrams as the user language as well as in ontologized Abstract State Machines as the formal model for DIP behavior interface descriptions. The following summarizes the usage scenario setting:

- The user Web service outputs a specification of the product it wishes to purchase and the the destination to which it should be delivered; it receives an offer providing the terms for purchase and delivery and returns an acceptance or rejection of the offer; and expects a confirmation of an acceptance
- The producer Web service receives a specification for a product and returns an offer for supplying it. It then receives a positive or negative answer to its offer, and confirms any positive answer
- The shipper Web service receives a specification for a package to deliver and its delivery address and returns an offer for delivery. It then receives a positive or negative answer to its offer, and confirms any positive answer.

4.2 Choreography Interface Descriptions with UML2 Activity Diagrams

We provide here two semantically-equivalent versions of the shipper choreography, the difference being in the data ontology used to answer the offer. In the first one as depicted in Figure 4.1, the user can either accept or decline the offer as two different messages. In the second one depicted in Figure 4.2, the answer is packed into one message containing either a positive or a negative response. In both cases, the choreography gives enough information in order to reason about what the outcome will be, depending on the answer.
Figure 4.1: Shipper Choreography - Unpacked Answer

Figure 4.2: Shipper Choreography - Packed Answer
4.3 Choreography Interface Descriptions with Ontologized ASMs

The sample is based upon the request for the delivery of an item from a supplier to a customer. The example here is of a choreography that corresponds to the Orchestration example in deliverable 3.4. In this example the answer is packed into one message. This example do NOT validate, as it appears that the validators have NO format for Choreography which they accept. The WSML definition puts on the proper method to define Choreography. Thus, no syntax for their definition fits the language definition.

```xml
wsmlVariant "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {"http://ontologies.deri.org/spChoreographies#", sp
  "http://ontologies.deri.org/supplierProvider#", spws
  "http://ontologies.deri.org/supplierProviderWebService#", dates
  "http://ontologies.deri.org/notableDates#", corp
  "http://ontologies.deri.org/corporate#" }

/*
* There is no format for definition of choreography that
* passes the validator. The following form is used.
*/

webService "http://ontologies.deri.org/spWebService"
    // Supplier-Producer Choreographies:
    interface spUserInterface
      choreography spUserChoreography
        stateSignature
          importsOntology ("http://ontologies.deri.org/SPOntology",
            "http://ontologies.deri.org/SPWSOntology"
          )
          in {
            sp#globalOffer withGrounding
            spWS#wsdl.interfaceMessageReference(SP_ServicePortType/GlobalOffer/In),
            sp#globalConfirm withGrounding
            spWS#wsdl.interfaceMessageReference(SP_ServicePortType/GlobalConfirm/In)
          }
          out {
            sp#productName withGrounding
            spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ProductName/Out),
            sp#destination withGrounding
            spWS#wsdl.interfaceMessageReference(SP_ServicePortType/Destination/Out),
            sp#globalAnswer withGrounding
            spWS#wsdl.interfaceMessageReference(SP_ServicePortType/GlobalAnswer/Out)
          }

  /* ProductName outputs a product name and destination given a
  requirement for a quantity of that product by a department.

  Destination outputs an address to which a product is to be shipped.
  The address of the department and the secretary's phone number are
  included in the destination specification.

  GlobalOffer inputs an offer for the sale and shipping of some
  product) and outputs an answer which is either a rejection of the
```
offer, or an acceptance with extra details possibly filled in, in return.

GlobalConfirm inputs a confirmation of a contract for purchase and shipping of some product, including all details of the transaction, and outputs the information to other programs in the user system as appropriate in response.

guardedTransition SupplierShipperRules /*
* Specify product and shipping destination
*/
  product hasValue ?pName, 
  quantity hasValue ?num, 
  dept hasValue ?dept] memberOf corp#requirement)
and
?dept[ contactSecretary hasValue ?sec 
  address hasValue ?deptAddress ] memberOf corp#department
and
?sec[mainOfficePhone hasValue ?secPhone, 
  name hasValue ?secName] memberOf corp#employee
do
add(_#[name hasValue ?pName, 
  quantity hasValue ?num] memberOf sp#productName)
and
add(_#[address hasValue ?deptAddress, 
  phone hasValue ?secPhone, 
  consignee hasValue ?secName] memberOf sp#destination)
endforall

/*
* Receive offer and return answer
*/
doforall {?goffer, ?delivTime, ?price, ?id, ?product, ?restriction, 
  ?priceLimit} with ( ?goffer[ deliveryTime hasValue ?delivTime, 
    totalPrice hasValue ?price, 
    quoteID hasValue ?id, 
    productType hasValue ?product] memberOf sp#globalOffer
  // ?selection, above, is the input. The items below are already
  // in the knowledge base.
  and
  ?restriction[maxPrice hasValue ?priceLimit, 
    productType hasValue ?product] memberOf sp#maxPrice 
  and
  before(?delivTime, dates#newYears2006)
  and
  wsm1#lessThan(?price, ?priceLimit)
do
add(_#[ deliveryTime hasValue ?delivTime, 
    totalPrice hasValue ?price, 
    quoteID hasValue ?id, 
    acceptance hasValue sp#accept] memberOf sp#globalAnswer)
endforall
/* * Receive confirmation and update own answer */
?gconfirm[
  quoteID hasValue ?id,
  pConfirm hasValue ?pConfirm,
  sConfirm hasValue ?sConfirm] memberOf sp#GlobalConfirm
and
?ganswer[
  quoteID hasValue ?id,
  acceptance hasValue sp#accept] memberOf sp#globalAnswer
do
update( ?ganswer[acceptance hasValue sp#confirmed])
endforall

interface spProducerInterface
  choreography spProducerChoreography
  stateSignature
    importsOntology _"http://ontologies.deri.org/SPOntology"
in {
  sp#producerRequest withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ProducerRequest/In),
  sp#producerAnswer withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ProducerAnswer/In)
}
out {
  sp#producerOrder withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ProducerOrder/Out),
  sp#producerConfirm withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ProducerConfirm/Out)
}

/* ProducerRequest inputs a request for purchase including product name and quantity. It outputs a ProducerOffer including details such as price, size, date available, etc. ProducerAnswer inputs an answer which is either a rejection of the offer, or an acceptance with extra details possibly filled in, in return. It outputs a GlobalConfirm which is a confirmation of a contract for purchase with values left optional, filled in, given any new data provided in the ProducerAnswer and internal considerations. */
guardedTransition ProducerRules

/*
 * Produce offer by Producer
 */
  ?request[ product hasValue ?product,
            quantity hasValue ?num ] memberOf sp#producerRequest
  and
  ?item[ product hasValue ?product,
         minQuantity hasValue ?min,
         maxQuantity hasValue ?max,
         price hasValue ?price ] memberOf corp#priceListItem
  and
  lessThan(?min, ?num)
  and
  lessThan(?num, ?max)
// leaving out calculation for when it will be ready and shipping size
// and generating quoteID
  do
    add( _#[ product hasValue ?product,
             quantity hasValue ?num,
             price hasValue ?price,
             size hasValue ?size,
             availableDate hasValue ?availableDate,
             quoteID hasValue ?quoteID ] memberOf sp#producerOffer )
  endForAll

/*
 * Producer receive answers and confirms if accepted
 */
doForAll {?poffer, ?price, ?id, ?product, ?restriction, ?priceLimit} with {
  ?answer[ pPrice hasValue ?price,
           productType hasValue ?product,
           quantity hasValue ?num,
           quoteID hasValue ?id,
           acceptance hasValue accept ] memberOf sp#producerAnswer
  and
  // ?answer, above, is the input. The offer below is already
  // in the knowledge base.
  ?poffer[ pPrice hasValue ?price,
           productType hasValue ?product,
           quantity hasValue ?num,
           quoteID hasValue ?id ] memberOf sp#producerOffer
  do
    add( _#[ pPrice hasValue ?price,
             quoteID hasValue ?id,
             productType hasValue ?product ] memberOf sp#producerConfirm)
  // other details would probably be added
  endForAll

/*
 * *
 */
interface spShipperInterface
choreography spShipperChoreography
stateSignature
importsOntology _"http://ontologies.deri.org/SPOntology"

in {
    sp#producerRequest withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ShipperRequest/In),
    sp#producerAnswer withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ShipperAnswer/In)
}

out {
    sp#producerOrder withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ShipperOrder/Out),
    sp#producerConfirm withGrounding
    spWS#wsdl.interfaceMessageReference(SP_ServicePortType/ShipperConfirm/Out)
}

/* ShipperRequest inputs a request for shipping including product type, size, and destination. It outputs a ShipperOffer including details such as price, delivery date, etc.

ShipperAnswer inputs an answer which is either a rejection of the offer, or an acceptance with extra details possibly filled in, in return. It outputs a ShipperConfirm which is a confirmation of a contract for shipping with values which were left optional filled in, given any new data provided in the ShipperAnswer and internal considerations.
*/

guardedTransition ShipperRules

/*
* Produce offer by Shipper
*/

forAll {?request, ?product, ?size, ?item, ?min, ?max, ?price, ?deliveryDate, ?quoteID } with (?request[ product hasValue ?product,
    size hasValue ?size ] memberOf sp#shipperRequest
and
    ?item[ product hasValue ?product,
    minSize hasValue ?min,
    maxSize hasValue ?max,
    price hasValue ?price ] memberOf corp#priceListItem
and
    lessThan(?min, ?size)
and
    lessThan(?size, ?max)
// leaving out calculation for delivery date and quoteID
do
    add( _#[ product hasValue ?product,
    size hasValue ?size,
    price hasValue ?price,
    deliveryDate hasValue ?deliveryDate,

quoteID hasValue ?quoteID ] memberOf sp#shipperOffer )
endForAll

/*
* Shipper receives answers and confirms if accepted
*/
?answer[ pPrice hasValue ?price,
productType hasValue ?product,
size hasValue ?size,
quoteID hasValue ?id,
acceptance hasValue accept] memberOf sp#shipperAnswer
and
// ?answer, above, is the input. The offer below is already
// in the knowledge base.
?poffer[ pPrice hasValue ?price,
productType hasValue ?product,
size hasValue ?size,
quoteID hasValue ?id] memberOf sp#shipperOffer
do
add(_#[pPrice hasValue ?price,
quoteID hasValue ?id,
productType hasValue ?product] memberOf sp#shipperConfirm)
// other details would probably be added
endForAll

The UML “packed answer” diagrams were used to generate the orchestration and
choreography examples. The communication of each service is described in its own
choreography. The orchestration invokes each service when the first data for it is ready,
composes data for each service when the input it expects is different from that which
is produced by one or more other services, advances its own state, and generates the
final goal success or failure.

Three basic services are depicted in the UML diagram, Figure 5.1. The chore-
ography for each was designed by expressing its inputs and outputs in WSML with
guarded transitions specifying outputs generated after inputs upon which they wait
are received.

The user service outputs a product name followed by a destination according to
the diagram. [Presumably the quantity of product is included with the product name
and the name of the party to receive the product and to be billed is included in the
destination, but this is not stated in the diagram.] The user then inputs a global offer
and outputs a global answer. After this the user service terminates after receiving
offer confirmation details.

The producer service inputs a producer request and outputs a producer offer. It
then inputs a producer answer and, if the answer is positive, outputs a producer
confirmation.

Similarly, the shipper service inputs a shipper request and outputs a shipper offer.
It then inputs a shipper answer and, if the answer is positive, outputs a shipper
confirmation.

* Upon receipt of a Product Name from a user, the producer service is invoked and
a producer request is prepared. Because a producer request would normally specify
the quantity of product as well as the name of the product, those two components
are assumed to be able to be extracted from the Product Name message sent by the user and are incorporated into the generated Producer Request. * Once a Producer Offer and a destination are received, a shipper is invoked. Although not included in the diagram, the assumption is that the request has been accepted. This is shown in the orchestration by a flag in the Producer Offer. The change in state once the shipper is invoked is signaled by modifying the request status flag from “accepted” to “processed”. * Once a Shipper Offer has been received, it is merged with the producer offer to produce a Global Offer, which the user choreography can input. Again, the assumption that the shipper has not rejected the request is made explicit. * Once a Global Answer is received from the user, both a Producer Answer and a Shipper Answer are extracted so that the appropriate services’ choreographies can handle them. * If a Global Answer is a rejection, the Composite Goal is designated a failure. * If a Global Answer is an acceptance and confirmations are received from both the shipper and producer, a Global Confirmation is created for the user and the Composite Goal is designated a success.

Two additional transitions were added to the choreography to cover the cases of rejection by the producer and shipper which were not included in the UML diagram:
* If the producer rejects the the Producer Request, a rejection is prepared as the “Global Offer” that will be returned to the user and the Composite Goal is designated a failure. In this case, the shipper is never invoked and no Global Answer is expected from the user. * If the shipper rejects the Shipper Request, rejections are prepared as the Global Offer that will be returned to the user and the Producer Offer that will be returned to the producer. The Composite Goal is designated a failure and no Global Answer is expected from the user.
5 Conclusions

This deliverable has provided the conceptual specification for choreography interfaces in DIP. A choreography interface is defined as:

*An choreography interface is a mandatory description element that specifies the behavior of a Web service or a Web service request for service consumption by describing the external visible communication structure (i.e. communicative acts to be interchanged and the process of this) along with a grounding towards Web service communication technology.*

The main aspects of this deliverable are:

- explication of the underlying model by understanding the behavior interfaces of Web services and requests as the main objects of interest for realizing automated Web service consumption and interaction
- identification of the requirements for choreography interface descriptions
- explication and rationale of the approach developed for choreography-related descriptions of Semantic Web services
- detailed discussion of *external visible behavior for service consumption, communication structure, and grounding* as the central conceptual elements of choreography interface descriptions
- differentiation and specification of service and client choreography interfaces
- a comprehensive example for definition of choreography interface with the developed description language.

The deliverable is accompanied by the document “DIP Interface Description Ontology” that specifies the common description language for choreography and orchestration interfaces.
REFERENCES


[34] WS-CDL working group. Web services choreography description language version 1.0, 2005.