DIP

Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP3: Service Ontologies and Service Description

D3.4

An Orchestration and Business Process Ontology

Sami Bhiri
Doug Foxvog
Stefania Galizia
Edward Kilgarriff
Barry Norton
Michael Stollberg
Laurentiu Vasiliu

January 20, 2006

Summary

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. 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

IST Project Number	FP6 – 507483	**Acronym**	DIP
Full Title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://dip.semanticweb.org/		
Document URL			
EU Project Officer	Kai Tullius		

Deliverable	**Number**	3.4	**Title**	An Orchestration and Business Process Ontology
Work Package	**Number**	3	**Title**	Service Ontologies and Service Description

Date of Delivery	**Contractual**	M24	**Actual**	Nov-05
Status	version 0.3		final ☒	
Nature	prototype ☐ report ☐ dissemination ☐ ontology ☒			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	see title page			
Resp. Author	Laurentiu Vasiliu		**E-mail**	laurentiu.vasiliu@deri.org
	Partner	NUIG	**Phone**	+353 (91) 495008

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	Service Composition, Orchestration, ASM, UML2AD, Workflow

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
15-09-2004	1	Laurentiu Vasiliu, Emilia Cimpian, Jakeck Kopecy	Based on the first deliverable structure, sections 1, 2, 3 are approached.
20-03-2005	2	Laurentiu Vasiliu, Emilia Cimpian, Jakeck Kopecy	Section 1, 2 3 refined; work in progress in coordination with WSMO working group. The necessity to define business processes more clear is issued; choreography and orchestration concepts are introduced as separate concepts.
28-05-2005	3	Laurentiu Vasiliu	Deliverable renamed to "An orchestration and business process ontology"; new deliverable structure proposed.
20-08-2005	4	Edward Kilgarriff	Redesigned the deliverable template and updated based upon the Lausanne meeting.
20-09-2005	5	Edward Kilgarriff	Revision draft delivered
25-10-05	6	Edward Kilgarriff	Adding structure to LaTeX template
26-10-05	7	Barry Norton	Refined structure
3-11-05	8	Barry Norton	Refined structure further after first draft
23-11-05	9	Michael Stollberg	Update to new deliverable structure
03-12-05	10	Doug Foxvog, Edward Kilgarriff	Added example and descriptions
04-12-05	11	Stefania Galizia	Removed Related Work and added Intended Tooling Support
05-12-05	12	Barry Norton	Rewrote sections 2 and 3
19-10-06	13	Stefania Galizia	Updated template

Reviewers				
	Jos de Bruijn		**E-mail**	jos.debruijn@deri.org
	Partner	UIBK	**Phone**	+43 (512) 507-6485
	Thomas Haselwanter		**E-mail**	thomas.haselwanter@deri.org
	Partner	UIBK	**Phone**	+43 (512) 507-6485

Project Consortium Information

Partner	Acronym	Contact
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	Berelcon	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 Féderale 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

Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK	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	ILOG	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG	inubit	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.	iSOCO	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	NIWA	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	OU	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	SAP	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.	Sirma	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 Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111
Vrije Universiteit Brussel	VUB Vrije Universiteit Brussel	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

This deliverable aims to provide a first conceptualisation and basic platform for an orchestration and business process ontology. 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. [5] Web services are designed having behind them business processes that have access to the exterior world by the web services interfaces. Web services describe themselves to the World Wide Web by exposing their capabilities, described in a semantic manner. In order to compose several web services, a clear conceptualisation of the formalism and implementation for achieving the desired results is needed. The work done in this deliverable focuses on the conceptualisation and formalism side of composing web services that are semantically described. The semantic description formalism used is WSMO and the semantic language considered is WSML.

1.1 Deliverable approach

Proposed Approach
The approach taken within the DIP consortium is a combination of WSMO/WSML Abstract State Machines (ASM) (see details in DIO appendix) meta-model approach and an ILOG composer approach. This joint approach has been developed using the DIP case study needs and the practical challenges to design and create an orchestration to be executed in semantic web services platforms (as WSMX and IRSIII).

Motivation
WSMO/WSML ASM based ontology modelling has been used because it is considered to be the minimal approach to modelling orchestrations and it allows the states that business processes need to go through to be captured in an ontological manner. However, a high level modelling approach is also needed to allow the user to easily define and generate orchestrations. Therefore, the ILOG composer approach provides a UML2 graphical interface along with UML2 to ASM mappings that allows translation from UML2 to ASM and WSML ASM. This approach has been also designed taking into account the prototype needs in DIP WP6 and both of the WSMO compatible execution environments that are used in DIP: WSMX and IRSIII. In this respect, while WSMO/WSML (ASM) formalism is executable by both execution environments, the ILOG composer will interconnect with WSMX and IRS III to provide orchestrations needed for the case studies implementations.

Strategy
Based upon the identified requirements, we define an approach for an orchestration interface description and properties, and based on the UML2AD-WSML mappings defined in DIO appendix we generate an example orchestration ready to be executed by WSMX / IRS III.
The contributions of the present work are 1) the realization of a joint methodology approach using WSMO/WSML (ASM) and the ILOG composer for creating generic orchestrations and 2) the creation of an example to be used by case studies partners to

generate their own orchestration. Our approach here is to allow the DIP case studies to build their own specific orchestration, allowing them to be flexible in building their own orchestration constructs. In the next section 1.2 the document organisation is described.

1.2 Organisation of the document

The present deliverable is structured as followings: after the introduction section, descriptions of requirements and approach are provided in section 2. There, the orchestration big picture is detailed together with the requirements for an Orchestration Interface descriptions as well as an approach for Orchestration interface description. Then, section 3 provides the ontology interface descriptions, focusing on Orchestration Interface properties, Orchestration types and intended tooling support. In section 4, a sample orchestration is provided to be used as an example for case studies and also as input for the orchestration component in WSMX for testing purposes. The case studies are expected to design their own orchestrations, based upon the example described in section 4 with the help of the available tools in DIP. Finally conclusions are issued and recommendation for future work. Specifically for the orchestration deliverable D3.4 (as well as the choreography deliverable D3.5) is the common description ontologies and language document attached as an appendix, called Dip Interface Description Ontology (DIO). In DIO, all language related issues necessary for orchestration but not necessarily related to the conceptual work are described. Also related work concerning orchestrations is listed in DIO, section 7.

2 Requirements and Approach

Orchestration provides a technique that allows service providers to realize the functionality of a Web service by composition of other Web services. A well-established mechanism to achieve service compositions in such contexts is workflow, as reviewed in the accompanying DIP Interface Description Ontology document.

The two major issues to be addressed in applying this work in DIP, where services are described in a WSMO compliant fashion, is what formalism we should like to use, and how to cope with specific characteristics of WSMO services. These latter consist particularly in the constraint of valid interactions with the services being composed to those governed by formally specified choreographies, and the requirement to support capability-based invocation.

2.1 The Big Picture

Orchestration provides a technique that allows service providers to realize the functionality of a Web service by composition of other Web services. An orchestration:

- describes those aspects of the internal business process of a Web Service where functionality of other Web services is utilized;
- decomposes the functionality into discrete parts;
- may attach existing services directly to some parts in order to achieve the decomposed functionality;
- may describe other parts of the decomposed functionality as *goals* to facilitate capability-driven execution at run-time;
- constrains the control flow between the decomposed parts of functionality in workflow style;
- defines the dataflow between the decomposed parts of functionality;
- denotes the multiple service interaction by one entity, which may be described to clients by its own choreography, abstracted from the orchestration description.

  This is illustrated in Figure 2.1.

On the other hand, a choreography for such a composite service does not detail the decomposition, and describes the client's point of view wherein the service is atomic and the only data-flow and control-flow is that which concerns interaction with the client. Hence by describing a choreography for the composite functionality, and defining an orchestration engine capable of executing it, a client can access this as they would any other service.

The orchestration continues as part of the service interface, however, as a client may carry out reasoning over this description. In particular, since we allow these design-time orchestrations to be under-specified via goals, they might like to know which such sub-goals have to be achieved, and under what constraints, for the service functionality to be provided. Furthermore the interface is also available to other services which may reuse the orchestration within their own definition.

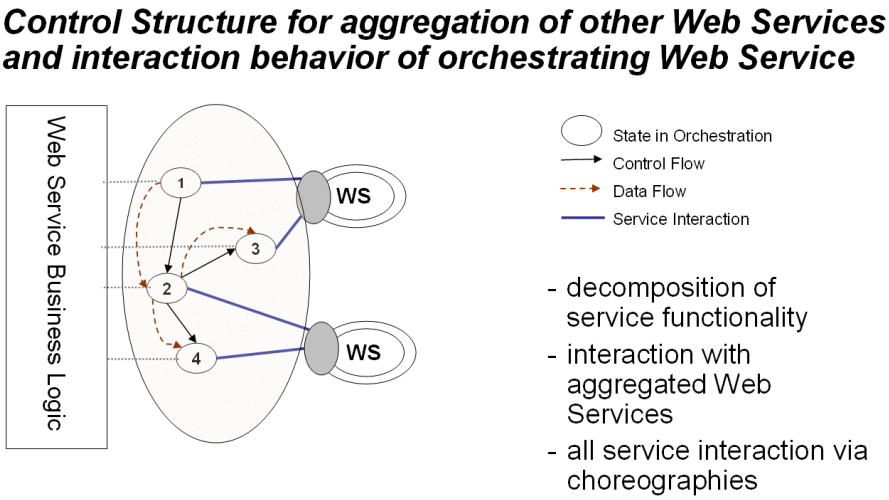

Figure 2.1: Orchestration Overview

2.2 Requirements for Orchestration Interface Descriptions

While WSMO has defined that orchestrations should be described formally as ASMs — and can be interpreted as implicitly requiring that goals, and not just services, must be orchestrated over — there are several more specific requirements that result from our detailed investigation in this area, and from the specific orientation of DIP. Each of the following requiments is introduced with its motivation:

1. **Two species of Orchestration**
   In order to facilitate the inclusion of goals in orchestrations, we distinguish two separate types of these:

   - **Design-time orchestration**: may nominate its component parts, and specify the message exchange and propagation of control between, both goals and specific identified services. Any orchestration attached to a service interface is, by definition, a design-time orchestration.
   - **Execution-time orchestration**: may nominate only identified services, and specifies data and control flow in the same manner. An execution-time orchestration is formed in the orchestration engine to be executed. This may involve discovery of services to meet goals (but if no tasks are specified as goals, it may merely be identical with the design-time orchestration).

2. **Workflow-like features**
   In order to design intelligible and efficient compositions of services, interaction with which may be long-lived, it is necessary that we support advanced control flow features. In particular, it is not appropriate to build only on sequential composition and choice, and must allow concurrency features common in workflow.

3. **Workflow Patterns**
   In order to facilitate communication with the research community, where our approach will be compared to others, it is important that we can express our workflows in terms of the common vocabulary of Workflow Patterns [12], and know how expressive our orchestration language is in these terms.

4. **Graphical Representation**
   In order to facilitate communication with the software engineering community, who will ultimately form a large proportion of the service providers under our architecture, it is important that we can express out workflows in a familiar and visual manner.

5. **Ontologies as Data Model**
   In order to be dealing with services that are specifically semantic web services, it is fundamental that ontologies are used as the underlying data model (see [4], [8]. This means that all resource descriptions are to be based on ontologies, and all data elements interchanged between client and Web service as well as between Web services are to be ontology instances. Thereby, support for the Semantic Web is assured inherently, and the basis for semantic interoperability as well as advanced information processing is given [3].

6. **Formal Description**
   In order to enable advanced, inference-based mechanisms for automating the Web service usage process, we require formal behavioural semantics for orchestration. Taking into account both choreography and orchestration, one central reasoning task within our approach, commonly referred to as conformance testing, 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 [11]. Another reasoning task is concerned with resolving process level mismatches that hamper successful interaction [1]. To formalise both conformance and safety properties it is required first to have a formal description of behaviour in which state, and potential to communicate at each state, can be judged.

   Such techniques provide the actual benefit of semantic behavioral descriptions of Web services, as syntactic service descriptions do not support such reasoning tasks. 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 dynamics that can occur in Web service interface descriptions, and it should integrate ontologies as the data model for the information that are to be interchanged.

2.3 Approach for Orchestration Interface Descriptions

While we discuss orchestration interface descriptions and respective modeling constructs 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 2.2.

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

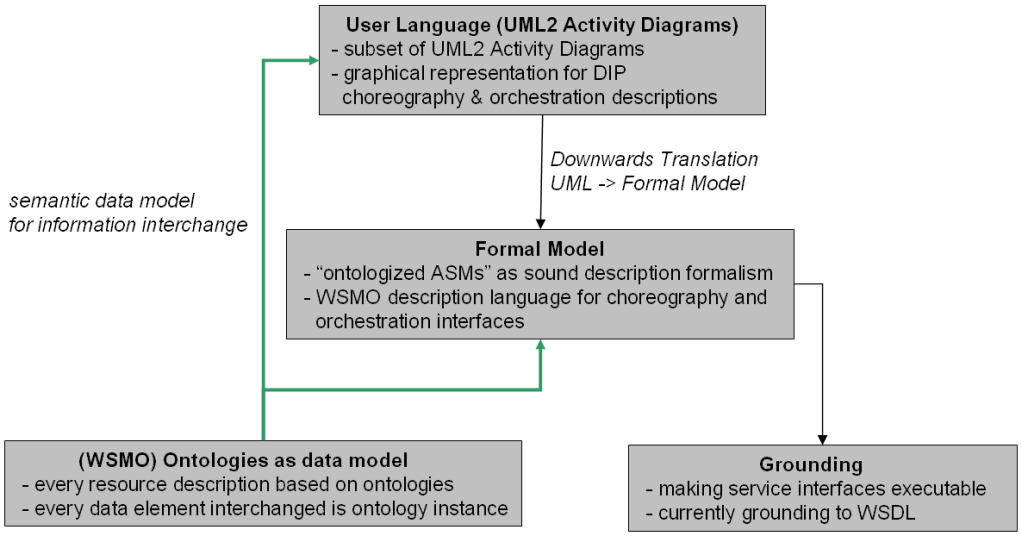

Figure 2.2: 2-Layered Description Languages for Choreography and Orchestration Interfaces in DIP

definition. Apart from being a graphical user language that is of significant familiarity to the engineering community, UML2 Activity Diagrams support many of the major Workflow Patterns as required [13].

As the formal model for semantically describing orchestration interfaces, we use so-called *ontologized Abstract State Machines* as the approach developed within WSMO for semantically describing the dynamics of Web service interface definitions [10]. 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 respective execution technologies for client-service as well as service-service interaction.

It is to be noted that this general language structure is the same used for describing choreographies. Thereby, the orchestration description languages extend the one for choreography interfaces with additional constructs (see DIP deliverable D3.5 "An Ontology for Web Service Choreographies").

3 Orchestration Interface Descriptions

On basis of the preceding examinations, this section explains conceptual aspects of orchestration interface descriptions in DIP. We first explain and rationalize the relevant aspects of orchestration interface descriptions in detail, concentrating on the distinction between design-time and execution-time orchestrations, then outline how orchestration descriptions can be created, and finally outline the intended tooling support for orchestration-related aspects in DIP.

3.1 Orchestration Interface Properties

In general, we say that DIP orchestrations have the following properties:

- **Decomposition of Web Service functionality**
  Through an orchestration we can nominate a set of components, the composition of which will lead to an intended functionality. As well as services and goals, we allow as first-class members of the workflow *mediators* between them. Whereas the interaction with goals and services is abstracted to message exchange, we view mediators as being atomic operations that can be achieved locally and in negligible time.

  In terms of message exchange, an orchestration can be viewed as an abstraction of a complex and interleaved set of message exchanges with multiple parties into a simpler exchange which a client can make with the composite service as a sole point of contact.

- **Control and Data Flow**
  While ASMs allow one to specify the structure of a computation in terms of information flow — and the ontologized version, as discussed in the "DIP Interface Description Ontology", allows one to specify this over an ontology — users or agents carrying out service composition will want to reason about the control flow and dataflow over the high-level tasks, rather than their realisation via low-level rules.

  For this reason, a higher-level model is supported from which we can translate down to ontologized ASMs. The use of first class mediators between message exchanges is particularly useful in this regard, being a basis for connection at a level removed from message exchange.

- **Graphical Representation**
  The UML is a very well-established formalism in the software engineering community, representing both the statics and dynamics of software from an object-oriented perspective. The UML Activity Diagrams language represents the dynamics of multi-party interaction.

  The ability to express a good proportion of Workflow Patterns [12], a common vocabulary giving semantics to the common features of a large survey of workflow languages, has recently been published [13].

  In this language we can express: mediators as actions, message exchange as signals, which specialise actions, dataflow and control flow as UML's Object Flow and control flow respectively.

3.2 Orchestration Types

In order to provide the benefits of capability-driven service invocation within the context of orchestration, it is necessary that we support the concept of goals. By allowing goals to be components within orchestrations, we allow the possibility that the discovery and selection of services to compose can be based on input data. Furthermore, we can account for a changing context of available services, effectively 'late-binding' the services based on those available at execution time.

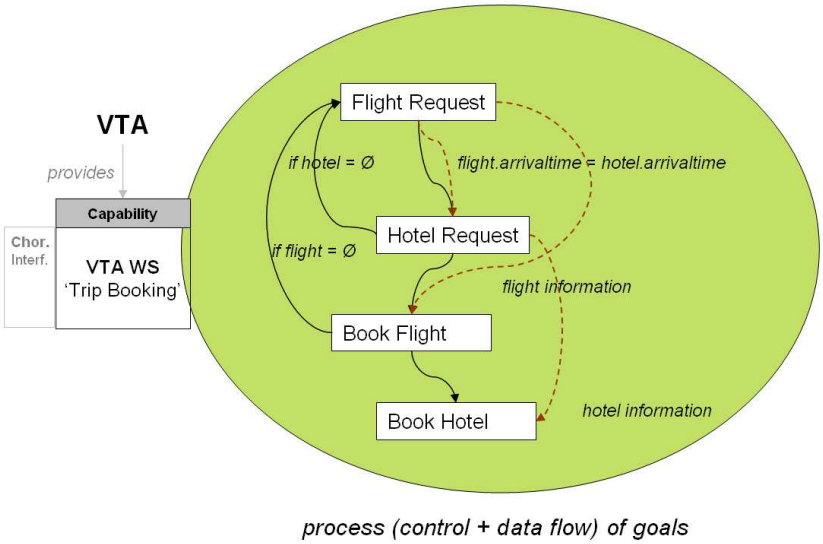

Figure 3.1: Abstract Orchestration as 'Process of Goals'

Figure 3.1 illustrates how we might build the 'virtual travel agency' example (VTA) as an orchestration over goals concerning locating and booking flights and hotels. Note that even if component services are not available to deal with the various goals in all geographical contexts, this design-time orchestration may still declare this capability and provide an abstract plan for achieving this, parameterised in such goals.

At execution time, a design-time orchestration that has no such parameterisation in goals may be taken directly as the execution-time orchestration and given to the orchestration engine for execution. A design-time orchestration that contains goals, however, uses those goals as the basis of discovery thus effecting on-the-fly service composition.

The general case is that both goals and mediation requirements must be bound at execution time. As well as data mediation requirements explicitly declared in the orchestration, it may be necessary to carry out process mediation between the interaction that has been declared and the choreography of each discovered service. Figure 3.2 illustrates how each message exchanged with a goal might map to multiple mesages. Furthermore, multiple goals may map onto the same under-lying service.

The intention in our definition of orchestration is that the execution will be governed by an orchestration engine that will execute the ASM which results from translating the execution-time orchestration. The initial step in this engine will involve the creation of an instance of the choreography engine for each component service. The message exchange declare in the orchestration will then be carried out against the relevant instance of the choreography engine.

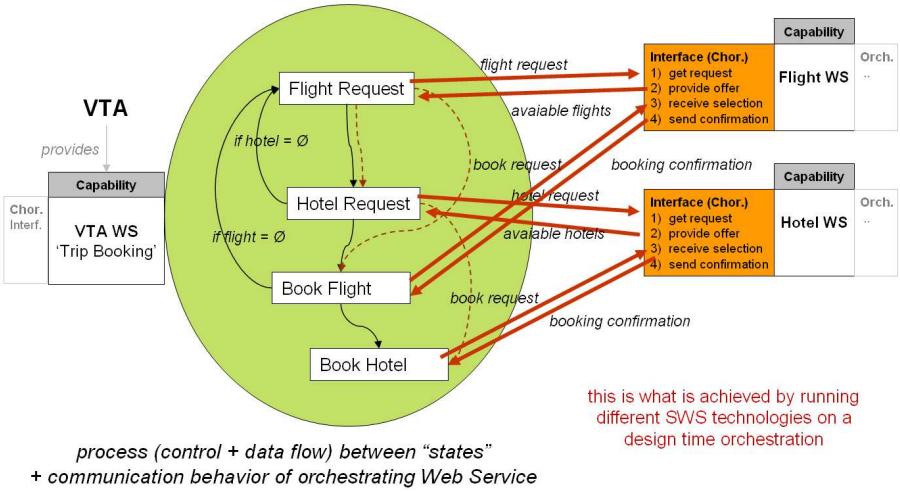

Figure 3.2: Executable Orchestration as 'Process of Operations')

3.3 Intended Tooling Support

3.3.1 WSMX

An orchestration engine, of the form explained above, will be developed within the WSMX environment during the deliverable D4a.20. This will utilise the existant WSMX Choreography Engine, and thereby Process Mediator, which enables interaction with a service by mediating between choreographies from a client and from a service point of view. One view of the client choreographies described in Deliverable D3.5 "An Ontology for Web Service Choreographies" is that these should be attached to the goals in design-time orchestrations to form such a client point of view.

3.3.2 IRS-III

Capability-driven invocation is one of the central principles of IRS, and the offering of pre-defined goals, backed this capability-driven brokerage of services to meet them, is its primary task. A related principle is that goals should strongly abstract from the interaction properties of services, and all goals are viewed as 'one-shot', *i.e.* processing a set of inputs to provide output as an atomic operation. The process mediation of external deployed services is also seen as a brokerage task, and a client choreography is a necessary part of all published services within the IRS.

Existing work on orchestration in IRS [2, 6] has considered the use of the OWL-S process model [7] as a workflow-oriented means for service composition. This is an appropriate model for the 'one-shot' approach to services and goals, while the notion of client choreography fills that gap between this view and services that have complex interaction demands expressed by their own service choreographies.

The new language Cashew, discussed as future work in "DIP Interface Description Ontology", will form an extension and rationalisation of this form of ontologized workflow-oriented model, and will be the basis of a full implementation in IRS, with a translation from the UML Activity Diagrams model being the input route for tools that deal directly with this model.

4 Example

This section provides an exemplary specification of orchestration 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 [9]. This has been chosen because the scenario covers both choreography and orchestration descriptions as well as the correlation between them. Besides, this use case is applied for illustrating the DIP Web service composition tool - which is strongly correlated to orchestration descriptions as the result of the composition procedure.

The following given a brief overview of the setting:

- The orchestration is the coordinator of the interaction between user, producer, and shipper Web services.
- The user Web service sees the orchestration as a single service from which it can purchase a product and have it delivered.
- Both the producer and shipper see the orchestration as a requester of its service: either the sale of a product or the delivery of some goods.

The orchestration descriptions are provided below, while the choreography interfaces are provided in DIP deliverable D3.5, "An Ontology for Web Service Choreography".

4.2 Orchestration Description in User Language (UML2AD)

This section includes two orchestration examples for the Shipper - Producer. Each one is based on a variant of the choreographies as defined in D3.5. In the first one, figure 4.1, the answer is packed into one message, whereas on figure 4.2, the answer is unpacked in two different messages whether it's a positive or a negative answer.

4.3 Orchestration Description in Formal Language (ASM)

This section includes the orchestration depicted in Figure 4.1 above expressed in ontologized Abstract State Machines. The example invokes the producer Web Service upon receiving a product specification from the user Web Service and invokes the shipper Web Service upon receipt of a destination from the user Web Service and product size included in a producer offer from the producer Web Service. It combines offers and confirmations from the shipper and producer Web Services into 'global' offers and confirmations, and unpacks a user answer into separate answers for the shipper and producer.

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

namespace {_"http://www.wsmo.org/ontologies/spChoreographies#", sp
_"http://www.wsmo.org/ontologies/supplierProvider#", corp
_"http://www.wsmo.org/ontologies/corporate#", pSWSDL
_"http://www.wsmo.org/ontologies/providerSupplierWSDL#" }

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

webService _"http://ontologies.deri.org/spWebService"
//  Supplier-Producer Choreographies:
interface spUserInterface orchestration productShippingOrchestration
  stateSignature productShippingOrchestrationSignature
  guardedTransitions producerShipperOrchestrationRules


stateSignature productShippingOrchestrationSignature

  in {
      sp#productName withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#productName/In),
      sp#destination  withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#destination/In),
      sp#shipperOffer withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#shipperOffer/In),
      sp#globalAnswer  withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#userAcknowledgement/In),
      sp#producerAnswer withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#producerAnswer/In),
      sp#shipperAnswer withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#shipperAnswer)
       sp#producerConfirmation withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#producerConfirmation/In),
      sp#shipperConfirmation withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#shipperConfirmation/In)

     }
  out {
      sp#producerRequest withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#producerRequest/Out),
      sp#globalOffer  withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#globalOffer/Out),
      sp#globalConfirmation withGrounding
          spWS#wsdl.interfaceMessageReference(
```

```
        SP_PortType/pSWSDL#globalConfirmation/Out),
      sp#compositeGoal  withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#compositeGoal/Out)
      }


  shared {
      sp#shipperRequest withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#pSRequest/Shared),

      sp#producerOffer withGrounding
          spWS#wsdl.interfaceMessageReference(
        SP_PortType/pSWSDL#producerOffer/Shared)
      }

guardedTransitions producerShipperOrchestrationRules

  if (?request[product hasValue ?product,
             quantity hasValue ?amt] memberOf sp#productName)
     then ((invoke sp#producerRequest)      // (Choreography uses ?request)
           and
           (add(_#[product hasValue ?product,
                   quantity hasValue ?amt] memberOf sp#producerRequest)))
       // other details may be generated

  if ((?pOffer[request hasValue ?request,
             acceptReject hasValue sp#acceptance,
               productSize hasValue ?pSize] memberOf sp#producerOffer) and
      (?location memberOf sp#destination))
     then ((invoke shipperRequest)
           and
           (update(?pOffer[acceptReject hasValue sp#processed])))

  if ((?pOffer[request hasValue ?request] memberOf sp#producerOffer) and
      (?sOffer[request hasValue ?request,
               acceptReject hasValue sp#acceptance] memberOf shipperOffer))
      then
        update(?request[acceptRejectOffer hasValue
                          _[producerOffer hasValue ?pOffer,
                            shipperOffer hasValue ?sOffer]
                           memberOf sp#globalOffer])

  if (?response[acceptReject hasValue ?userAcceptance,
                request hasValue ?request] memberOf sp#globalAnswer)
     then
 ((add(_[acceptance hasValue ?userAcceptance,
             request hasValue ?request] memberOf sp#producerAnswer)) and
     (add(_[acceptance hasValue ?userAcceptance,
                request hasValue ?request] memberOf sp#shipperAnswer)))
  // If the user acceptance has separate details for shipper and producer,
  // they should be separated here.

  if (?response[acceptReject hasValue sp#reject,
                request hasValue ?request] memberOf sp#globalAnswer)
     then
      (add(_[request hasValue ?request,
```

```
            succeed hasValue sp#failure] memberOf sp#compositeGoal))

  if ((?prodConf[request hasValue ?request] memberOf sp#producerConfirmation) and
      (?shipConf[request hasValue ?request] memberOf sp#shipperConfirmation) and
      (?response[acceptReject hasValue sp#accept,
                 request hasValue ?request] memberOf sp#globalAnswer))
    then
      ((add(_[request hasValue ?request,
               shipConfirmation hasValue ?shipConf,
               prodConfirmation hasValue ?prodConf]
              memberOf sp#globalConfirmation))
       and
       (add(_[request hasValue ?request,
              succeed hasValue sp#success] memberOf sp#compositeGoal)))

/// The following are not included in the diagrammed orchestration:

 if (?pOffer[request hasValue ?request,
             acceptReject hasValue sp#rejected] memberOf sp#producerOffer)
    then ((add(_[acceptReject hasValue sp#rejected] memberOf sp#globalOffer))
          and
          (add(_[request hasValue ?request,
             succeed hasValue sp#failure] memberOf sp#compositeGoal)))
  // If producer rejects the request, prepare a rejection for the user
  // as the Global Offer and define the goal as having failed.

  if (?sOffer[request hasValue ?request,
              acceptReject hasValue sp#rejected] memberOf sp#shipperOffer)
    then ((add(_[acceptReject hasValue sp#rejected] memberOf sp#globalOffer))
          and
          (add(_[request hasValue ?request,
                 acceptance hasValue sp#rejected] memberOf sp#producerAnswer))
        and
          (add(_[request hasValue ?request,
                 succeed hasValue sp#failure] memberOf sp#compositeGoal)))
  // If shipper rejects the request, prepare a rejection for the user
  // as the Global Offer and for the producer as a Producer Answer,
  // and define the goal as having failed.
```

The last two rules ('guarded transitions') in the orchestration cover the case of rejection of the product request or the shipper request by the producer or shipper respectively. In either case, the 'global offer' is a rejection and the overall goal fails. If the producer rejects the request, then no shipper request is required (or appropriate).

The orchestration invokes the other services under the appropriate conditions and calculates inputs needed by their choreographies if that data is not directly available.

The orchestration depicted in UML diagram Figure Ops Packed, was interpreted as follows:

* 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 faliure. * 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.

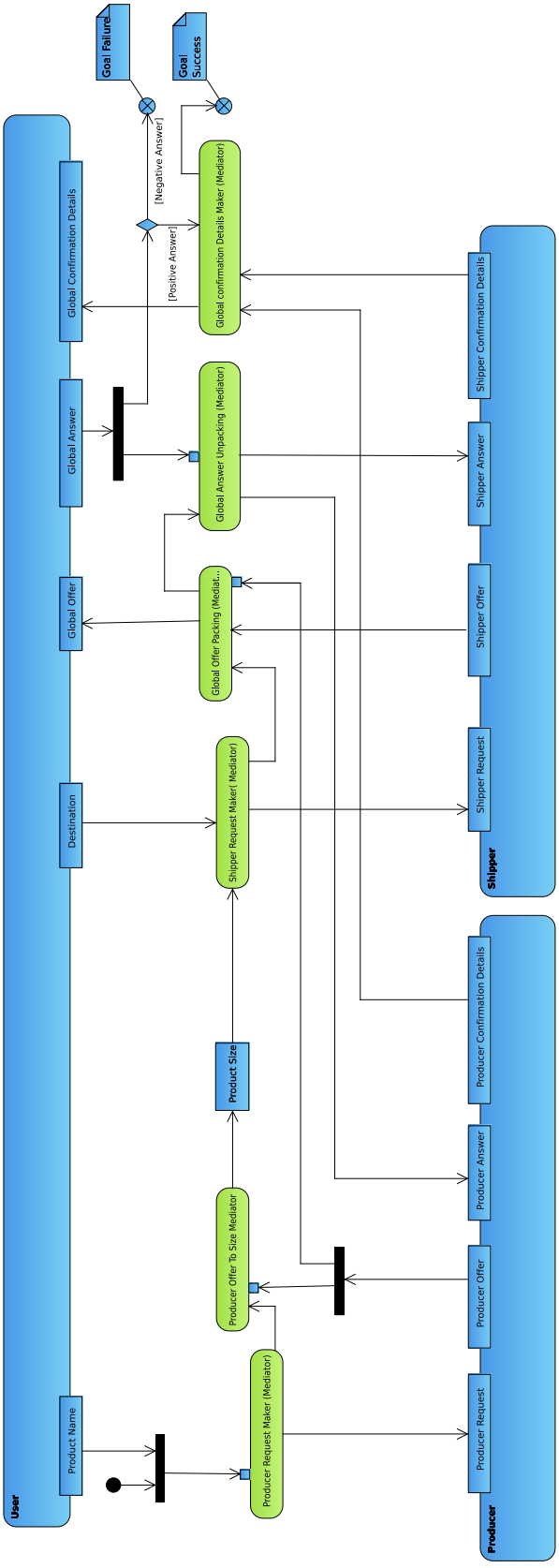

Figure 4.1: Shipper-Producer Orchestration in UML2AD (Packed Answer)

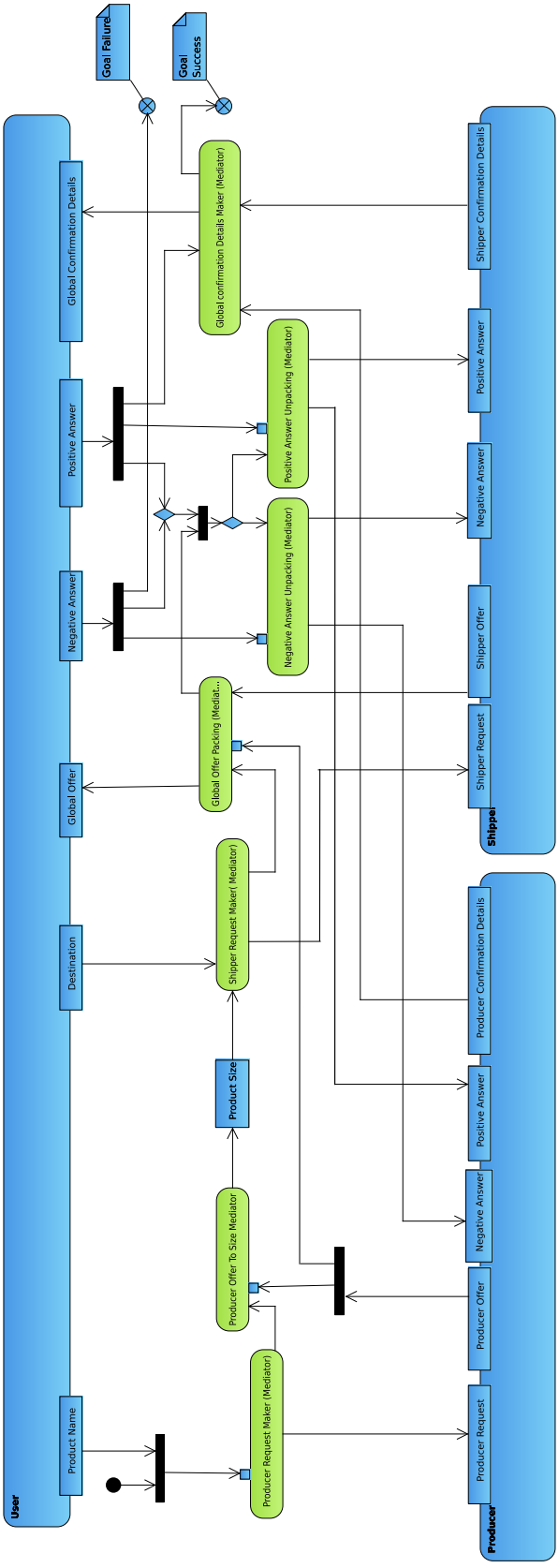

Figure 4.2: Shipper-Producer Orchestration in UML2AD (Unpacked Answer)

5 Conclusions

In this deliverable we described our vision on web services orchestration (composition) providing an overview of its internal structure. The requirements needed for an orchestration interface have been listed, differentiating between design time and execution time orchestration. This highlights the need for workflow-like features and patterns, emphasizing the importance of a graphical representation, ontologies usages and finally, the need for a well defined formal description. The approach for an orchestration interface description has been discussed, with emphasis placed on web services decomposition functionality, control and data flow and also graphical representation. The intended tooling support has been presented, providing details over the two WSMO reference implementations that are WSMX and IRS III and in particular over their current and future orchestration implementation approaches. An example (shipper-producer usage scenario) has been provided, showing both the UML2AD orchestration representation and the WSML ASM formal language description of the example. All language related aspects and related work have been described in the DIP interface Description Ontology (DIO) document, attached to this deliverable.

References

[1] E. Cimpian and A. Mocan. WSMX Process Mediation Based on Choreographies. In *Proceedings of the 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management at the BPM 2005, Nancy, France*, 2005.

[2] R. Confalonieri, J. Domingue, and E. Motta. Orchestration of semantic web services in IRS-III. In *Proceedings of the First AKT Workshop on Semantic Web Services (AKT-SWS04)*, volume 122. CEUR Workshop Proceedings, 2004.

[3] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and E-Commerce*. Springer, Berlin, Heidelberg, 2 edition, 2003.

[4] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

[5] H. Haas and A. Brown. Web Services Glossary. W3C Working Group Note 11 February, 2004.

[6] F. Hakimpour, J. Domingue, E. Motta, L. Cabral, and Y. Lei. Integration of OWL-S into IRS-III. In *Proceedings of the First AKT Workshop on Semantic Web Services (AKT-SWS04)*, volume 122. CEUR Workshop Proceedings, 2004.

[7] D. Martin, editor. *OWL-S 1.1 Release*. 2004. http://www.daml.org/services/owl-s/1.1/.

[8] S. McIlraith, T. Cao Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.

[9] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and Monitoring Web Service Composition. In *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services held in conjunction with ICAPS 2004, Whistler, British Columbia, Canada, June 3-7*, 2004.

[10] J. Scicluna, A. Polleres, and D. Roman (eds.). Ontology-based Choreography and Orchestration of WSMO Services. Deliverable D14, 2005. available at: http://www.wsmo.org/TR/d14/.

[11] M. Stollberg. Reasoning Tasks and Mediation on Choreography and Orchestration in WSMO. In *Proceedings of the 2nd International WSMO Implementation Workshop (WIW 2005), Innsbruck, Austria*, 2005.

[12] W. M. P. van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Systems*, 14(1):5–51, 2003.

[13] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based analysis of UML activity diagrams. BETA Working Paper Series WP 129, University of Technology, 2004.