DIP

Data, Information and Process Integration with Semantic Web Services

FP6 - 507483

Deliverable

WP 6: Architecture & Interoperability

D6.12

Execution Semantics for Semantics-Enabled Systems, Standardisation activity with OASIS

John Domingue

Barry Norton

Omair Shafiq

Maciej Zaremba

Brahmananda Sapkota

Matthew Moran

June 30th, 2006

11/07/2006

SUMMARY

The DIP Architecture strives to provide guidelines for a Semantic Web Services execution platform enabling dynamic discovery, mediation and invocation of Semantic Web Services. It is an architecture based on loosely-coupled services following the principles of Service Oriented Architecture where interactions between the services are not by any means predefined but can be specified during the system exploitation.

The content of this deliverable is also that of the emerging deliverable of the OASIS Technical Committee for Semantic Execution Environments (SEE TC) titled "SEE Execution Semantics". This OASIS deliverable is being driven by members of the DIP consortium and is still in an initial state. We deliberately limit this DIP deliverable so that it is aligned with the, albeit early, work of the SEE TC.

Execution semantics are formal descriptions of the operational behaviour of the DIP architecture. By separating the description of the system behaviour from the system's implementation, we aim to achieve greater flexibility in how the implementations of the DIP architecture can be used and avoid the necessity to rebuild the system in the event of a new or changed requirement on how the system should operate.

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**	6.12	**Title**	Execution Semantics for Semantics-Enabled Systems - Standardisation activities with OASIS and W3C
Work package	**Number**	6	**Title**	Architecture and Interoperability

Date of delivery	**Contractual**	M 30	**Actual**	17-June-06
Status	Version. 0.01		final	
Nature	Prototype ☐ Report ☑ Dissemination ☐ Ontology ☐			
Dissemination Level	Public ☑ Consortium ☐			

Authors (Partner)	Matthew Moran (NUIG), Brahmananda Sapkota (NUIG), John Domingue (OU), Barry Norton (OU), Omair Shafiq (UIBK), Maciej Zaremba (NUIG)			
Responsible Author	Matthew Moran		**Email**	Matthew.moran@Nuideri.org
	Partner	NUIG	**Phone**	00 353 91 495017

Abstract (for dissemination)	Execution semantics are formal descriptions of the operational behaviour of the DIP architecture. This deliverable defines the mandatory execution semantics for the DIP system using UML activity diagrams and proposes the use of the DIP Orchestration ontology as the means for providing the formal descriptions of the execution semantics for the DIP architecture using WSML.
Keywords	Execution semantics, behaviour, architecture

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
17-jun-06	001	Brahmananda Sapkota, Matthew Moran	Initial version based on the corresponding deliverable from the OASIS SEE Technical Committee sent for review.
30-jun-06	**002**	Matthew Moran	Update based on review comments

Reviewer	

	Adrian Mocan		**Email**	Adrian.mocan@deri.org
	Partner	UIBK	**Phone**	
	Silvestre Losada		**Email**	slosada@isoco.com
	Partner	ISOCO	**Phone**	

Project Consortium Information

Partner	Acronym	Contact
National University of Ireland Galway	NUIG	Dr. Sigurd Harand Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland Email: 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
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 Email : 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. Email: 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 Email: abecker@fzi.de Tel: +49 721 9654 0
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

Partner	Acronym	Contact
ILOG SA	ILOG	Christian de Sainte Marie 9 Rue de Verdun, 94253 Gentilly, France Email: csma@ilog.fr Tel: +33 1 49082981
inubit AG	Inubit	Torsten Schmale inubit AG Lützowstraße 105-106 D-10785 Berlin Germany Email: 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 Email: rbenjamins@isoco.com Tel. +34 913 349 797
MDR Partners	MDR	Rob Davies MDR Partners 8 St. Andrew Street Hertford, Herts. United Kingdom, SG14 1JA, Email: rob.davies@mdrpartners.com +44 (0)208 8763121
Hanival Internet Services GmbH	HANIVAL	Alexander Wahler Hanival Internet Services GmbH Kirchengasse 13/1a A-1070 Wien Email: wahler@niwa.at Tel:+43(0)1 3195843-11 |
The Open University	OU	Dr. John Domingue Knowledge Media Institute The Open University, Walton Hall Milton Keynes, MK7 6AA United Kingdom Email: 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 Email: elmar.dorner@sap.com Tel: +49 721 6902 31
Partner	**Acronym**	**Contact**

Sirma AI Ltd.	Sirma	Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse Sofia 1784, Bulgaria Email: 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 Email: 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 Email: Pieter.De.Leenheer@vub.ac.be Tel.: +32 (0) 2 629 3749

LIST OF KEY WORDS/ABBREVIATIONS

Keywords

Execution semantics, Process, Dynamic model, State Machine

Abbreviations

ASM	Abstract State Machines
BPEL	Business Process Execution Language
CASHEW	Composition And Semantic enHancement of Web Services
EAI	Enterprise Application Integration
OASIS	Organization for the Advancement of Structured Information Standards
SEE	Semantic Execution Environment
SOA	Service Oriented Architecture
SWS	Semantic Web Service
UDDI	Universal Discovery Description and Integration
UML	Unified Modeling Language
WS	Web Service
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Execution Environment

TABLE OF CONTENTS

SUMMARY I
LIST OF KEY WORDS/ABBREVIATIONS VII
TABLE OF CONTENTS VIII
1 INTRODUCTION 1
2 MOTIVATION AND RATIONALE 2
2.1 Motivation 2
2.2 Rationale behind the use of Execution Semantics 3
3 DESCRIPTION FORMALISMS 4
3.1 Activity Diagrams 4
3.2 Workflow – Cashew 4
3.3 Abstract State Machines – WSML 5
4 OVERALL APPROACH 6
4.1 Overview of the DIP Architecture 6
4.2 Goal Based Web Service Discovery 7
4.3 Goal Based Service Execution 8
4.4 The invocation of Web Services 10
4.5 Register Execution with DIP Architecture 11
5 THE DIP ORCHESTRATION ONTOLOGY AND EXECUTION SEMANTICS 11
6 SUMMARY 12
7 REFERENCES 12

LIST OF FIGURES

Figure 1: Capability based service execution 9
Figure 2: Goal centric Web Service Discovery in DIP 8
Figure 3: The invocation of Web Services 11

1 INTRODUCTION

The hot topic in today's design of software architectures is to satisfy increasing software complexity as well as new IT needs, such as the need to respond quickly to new requirements of businesses, the need to continually reduce the cost of IT or the ability to integrate legacy and new emerging business information systems. In the current IT enterprise settings, introducing a new product or service, integrating multiple services and systems can present unpredictable costs, delays and difficulty. Existing IT systems consist of a patchwork of legacy products, monolithic off-shelf applications and proprietary integration. It is often the reality that users on "spinning chairs" manually re-enter data from one system to another within the same organization. The past and existing efforts in the Enterprise Application Integration (EAI) have not yet presented a successful and flexible solution to these problems. EAI projects are often lengthy and often over budget.

Service Oriented Architecture (SOA) solutions are seen as the next evolutionary step for software design. SOA is a style of software system architecture in which components are defined as independent services with well-defined, invocable interfaces. SOA improves the chances of cost-effective integration and flexibility to business processes. Web service technology is a fundamental enabler for SOA through the WSDL, SOAP, UDDI specifications. The Business Process Execution Language (BPEL) allows composition of services into complex processes as well as their execution. Although Web services technologies around UDDI, SOAP and WSDL have added a new value to the current IT environments in regards to the integration of distributed software components using web standards, they cover mainly characteristics of syntactic interoperability. With respect to a large number of services which exist in IT environments based on SOA, the problems with service discovery or selection of the best services conforming to users' needs, as well as resolving heterogeneity in services' capabilities and interfaces remain lengthy and costly. For this reason, machine processable semantics should be used for description of services in order to allow total or partial automation of tasks such as discovery, selection, composition, mediation, invocation and monitoring of services.

Execution semantics are formal descriptions of the operational behaviour of the DIP architecture. By separating the description of the system behaviour from the system's implementation, we aim to achieve greater flexibility in how the implementations of the DIP architecture can be used and avoid the necessity to rebuild the system in the event of a new or changed requirement on how the system should operate.

In terms of the behaviour of the DIP architecture, we differentiate two types of services: middleware platform services and application services. Platform services are mandatory to enable the DIP infrastructure to deliver its functionality as defined in the execution semantics. User services are exposed by information system external to the DIP infrastructure and the role of the DIP platform, and its services, is to coordinate interactions between them. Execution semantics define the system behaviour in terms of the middleware platform services. They describe in a formal, unambiguous notation how the system operates. Once the platform services are specified, they can be combined in arbitrary ways. In this context execution semantics can be perceived as a layer on the top of platform services where the overall execution of SOA system for the given scenario can be specified by providing business logic that combines control and

data flow between the platform services. Services that are performing their tasks are completely unaware of this upper business layer and their role within it. We can distinguish two phases of our work where different formalisms can be applied:

- Reaching agreement on most required DIP behaviors. In is of utmost importance to reach a consensus on most fundamental behaviors that DIP should support and to present them by graphical means in human intelligible way that allows quickly grasping the essence of each of them.
- Expressing identified execution semantics in other formalisms (e.g. Cashew [7], Abstract State Machines [1], etc.). These formalisms do not necessarily have to use the graphical notation. execution semantics expressed in these other formalisms should unambiguously reflect the ones defined in the first phase and they should be done once the work in the first phase is completed to avoid going back-and-forth between them whilst agreeing on most necessary DIP behaviours.

The structure of this deliverable is as follows. First some background to execution semantics are provided, next three approaches to describing behaviour are introduced. Four mandatory execution semantics for the DIP architecture are defined and finally, the deliverable describes how the work carried out in work package 3 on an orchestration ontology provides the formal language which will be used in the M36 deliverable to describe the execution semantics.

2 MOTIVATION AND RATIONALE

2.1 Motivation

Specification of system behavior can be viewed as control and data flow between system components (services in SOA), where the actual actions take place. Developers tend to create architectures for specific, current needs which can result in rigid system behaviour. The DIP architecture takes a quite opposite approach, where system building blocks are well-defined and ready to be utilized in various scenarios, not necessarily considered at design time. The DIP architecture uses an event based messaging design to compose loosely coupled services. This leads to various possible execution semantics for the system since the activation of the services are stimulated by events as they occur and there are no fixed bindings between the platform services. Services can create or consume events but they cannot invoke each other directly. They can cooperate with each other on the interface level but they do not refer to each other implementation directly.

There are many methods to specify and further execute system behavior. Some of them allow the designer to specify the behaviour graphically, but in an often informal way (e.g. UML diagrams); some of them are formal and unambiguous yet require significant experience from the designer (e.g. Petri net). In this deliverable, we introduce a layered approach to the definition of execution semantics using a combination of UML (for modellers) and Abstract State Machines (for the formalism) and CASHEW (adding semantics) as the bridge between them. This approach is described in more detail in section 3.

2.2 Rationale behind the use of Execution Semantics

A software design process should result in a design that is both an adequate response to the user's requirements and an unambiguous model for the developers who will build actual software. A design therefore serves two purposes: both to guide the builder in the work of building the system and to certify that what will be built will satisfy the user's requirements.

Execution semantics, or operational semantics, is the formal definition of system behavior in terms of computational steps (*cf.* denotational and algebraic semantics, which concern *what* is computed, not *how* it is computed). As such, it describes in a formal, unambiguous language how the system operates. Since in a concurrent and distributed environment the meaning of the system, to the outside world, consists of its interactive behavior, this formal definition is called Execution semantics.

Major advantages of execution semantics over informal methods are the following:

- **Foundations for model testing.** It is highly desirable to perform simulation of the model before the actual system is created and enacted. It allows one to detect anomalies like: deadlock, livelock or tasks that are never reached. However, as pointed out by Dijkstra in [3] model simulation allows pointing out presence of errors, but not lack of them. Nevertheless, it is a paramount to detect at least some of system malfunctions during a design time instead of the run-time. Therefore, semantics of utilized notations has to be perfectly sound in order to create tools enabling simulation of created models. Only formal, mathematical foundations can meet these requirements.
- **Executable representation.** Similarly like in the case of model testing, using formal methods provides sound foundation to build an engine able to execute created models. Such an engine would not necessarily need to be able to detect livelock or other model faults. This distinguishes this bullet from the previous one.
- **Improved model understanding among humans.** Soundness of the utilized method significantly improves or even rules out ambiguities in model understanding by humans.

Several methods exist to model software behavior. Some of them model system behavior in a general way like UML diagrams, other impose more formal requirements on model, for instance Petri net-based methods, process algebra, modal and temporal logics and type theory. These methods have different characteristics: some are more expressive than others; some are more suited for a certain problem domain than others. Some methods provide graphical notation like UML or Petri nets, some are based on syntactic terms like process calculi and logics; some methods have tool support for modeling, verification, simulation or automatic code generation and others do not.

We impose two major requirements on the methodology utilized for modeling most fundamental DIP behavior. Firstly it has to use understandable and straightforward graphical notation, secondly it has to be unambiguous. These two requirements are met by UML Activity Diagrams, which are familiar to the engineering community and whose execution semantics can be disambiguated, for instance in the semantics specified by Eshuis [6].

3 DESCRIPTION FORMALISMS

This section describes the three-layer approach to the definition of execution semantics for the DIP architecture. Section 3.1 describes why UML is chosen as the modelling notation (layer 1). Cashew in section 3.2 is a language for workflow patterns that provides a semantics for UML activity diagrams. Abstract State Machines, described in section 3.3 are the formalism used by WSMO to describe dynamic aspects of Semantic Web services.

3.1 Activity Diagrams

UML 2.0 (Unified Modeling Language) is a widely accepted and applied graphical notation in software modeling. It compromises of a set of diagrams for system design capturing two major aspects, namely static and dynamic system properties. Static aspects specify system structure, its entities (objects or components) and dependencies between them. These structural and relational aspects are modeled by diagrams like: Class Diagrams, Component Diagrams and Deployment Diagrams. Dynamic aspects of the system are constituted by control and data flow within the entities, specified as: Sequence Diagrams, State Machine Diagrams and Activity Diagrams. Originally UML was created for modeling aspects of object-oriented programming (OOP). However, it has to be emphasized that UML is not only restricted to the usage in OOP area, but is also applied in other fields like for instance Business Process Modeling.

The primary goal of UML diagrams is to enable common comprehension of structure and behavior of modeled software among the involved parties (e.g. designers, developers, other stakeholders, etc.). To the detriment of UML notation, this information is conveyed in informal way that may lead to ambiguities in certain cases as pointed out in [6]. Since we want to model behavior of DIP in formal, unambiguous yet easily comprehensive manner thus appropriate modeling method has to be chosen.

Activity Diagrams depict a coordinated set of activities that capture the behavior of a modeled system. Activity Diagrams specify a control and data flow between the entities providing language constructs that enable to model elaborate cases like parallel execution of entities or flow synchronization.

Due to the wide proliferation of UML notation several efforts were carried out to concretize its execution semantics, especially regarding the dynamic aspects of UML notation. Execution semantics for UML Activity Diagrams, in the context of workflow modeling, was specified in [4], [5]. UML Activity Diagrams fulfill our requirements; therefore they are going to be used throughout this document, consistently with the semantics given by Eshuis in [4], to specify operational behavior of DIP.

3.2 Workflow – Cashew

In recent years, much attention has been paid to the role of the workflow style of specification in the definition of service composition. This is both a natural fit with the requirements for service composition, providing similar control and data flow idioms, and provides a useful synergy due to its existing application to the specification of Business Process Modeling. A consideration of the workflow style thus allows communication with the community we work within, providing links to approaches such as BPEL and OWL-S, and with one major community with whom we apply our work, many of our case studies being drawn from the B2B domain.

Workflow Patterns is the name given to the work of a community developing a common vocabulary and semantics to capture all variants of the workflow approach [8]. Although this work has a direct graphical representation via the language YAWL [13], and a formal semantics due to an extension of coloured Petri nets [12]. Cashew defines an ontology and language [7] for the orchestration of Web services. The consequent language is built around a vocabulary which can be diagramed in UML Activity and mapped through process algebra to the Abstract State Machine formalism used by WSMO to define service orchestrations. Furthermore it adapts and extends the OWL-S process model, aligning it with Workflow Patterns and WSMO, in order to achieve this.

By sticking to those Activity Diagrams that illustrate valid workflows in the Cashew language, it avoids the more esoteric features, and moreover arbitrary and esoteric combinations of features, that lead to semantic ambiguity in the literature with respect to UML Activity Diagrams. A definition of Cashew and a formal translation to UML Activity Diagrams is being defined within the DIP project as part of the work package 3 deliverable, D3.8 [14] (available initially only to DIP consortium members).

3.3 Abstract State Machines – WSML

Abstract State Machines (ASMs) are a development of automata theory and algebraic specification with the following defining properties:

- *Minimality*: ASMs provide a minimal set of modeling primitives, i.e., enforce minimal ontological commitments. Therefore, they do not introduce any ad-hoc elements that would be questionable to be included into a standard proposal.
- *Maximality*: ASMs are expressive enough to model any aspect around computation.
- *Formality*: ASMs provide a rigid framework to express dynamics.

Rather than working with a Petri net semantics for Activity Diagrams, such as that of [2], or of Workflow Patterns, we adopt ASMs as our low-level semantic model. This provides us with a number of advantages. Firstly, we inherit from WSMO an elegant means to combine the behavioural part of this formalism with the structural semantics of ontologies, via which the capabilities and requirements of semantic web services are expressed. Secondly, we inherit an executable framework for affecting such behaviour in terms of Semantic Web services via the DIP platforms.

The DIP project aims to provide two means by which the kind of models this document contains can be translated into a low-level semantics in ASMs. Firstly a direct translation from UML Activity Diagrams is defined. Secondly a more formal and compositional translation is provided via Cashew and in process algebraic style. These are out of the scope of this deliverable but are defined in deliverable 3.8 [14].

Additionally, it is proposed to formally specify the functional capabilities of the platform services themselves using WSML (Web Services Modeling Language) [2]. WSML is a language based on logic foundations providing semantic descriptions of various aspects related to Web services. Having semantic descriptions of the components one could combine them automatically or discover them. Formal description rules out the ambiguities and provides the foundations for reasoning. However, the question is the level of the granularity of semantic descriptions that the

components will be fitted with (i.e. what is the level of the detail captured by these descriptions).

4 OVERALL APPROACH

We define four mandatory execution semantics for the DIP architecture that are described in more detail in the following subsections. The public DIP deliverable, D6.11 [15] provides a description of the service oriented architecture designed for DIP which is based on independent, well-defined, platform services (discovery, mediation, orchestration etc.). It is on top of these platform services that the execution semantics are defined. They allow the most common execution scenarios related to the discovery, mediation and communication with Semantic Web services to be formally described outside the implementation of the system itself.

Four execution semantics are defined in the following sub-section. For each, the following notation is used, where *execSemName* stands for the name of the entry point to the system for the execution semantics, *input data types* are the types of the input parameters required to start the execution semantics and *return data type* is the data type returned if the execution semantics completes successfully.

execSemName* (*Input data types*): *Return data type

Section 4.1 provides an overview of the components of the DIP architecture used in the descriptions of the execution semantics. Section 4.2 describes the scenario where goal-based service discovery without service invocation is desired. Section 4.3 is an extension of that described in 4.2 – both service discovery and service invocation are required. The client's Goal and all data instances the client wishes to send to the potential provided service are specified together as input. In section 4.4, the situation described relates to where a client already knows the service they wish to invoke and have the required data for this invocation available. Finally, section 4.5 describes a simpler operational scenario where a client of the DIP system registers a description of a Semantic Web service with which they wish to communicate. This scenario establishes a context for message exchange between the client and this service.

4.1 Overview of the DIP Architecture

The DIP architecture is described in detail in the public DIP deliverable D6.11 [15]. As the names of the principle functional components are used in the descriptions of the following subsections, we list these components here for completeness along with brief descriptions.

Discovery. The Discovery component is concerned with finding Web service descriptions that match the goal specified by the service requester.

Non-functional Selector (Selection). The Non-functional Selector is used to select the most suitable service from a list of matching services matched by discovery. For example, if discovery results in more than one service that satisfies the goal, this component may be used to choose one based on specified preferences.

Data Mediator. The Data Mediator has the role of reconciling the *data* heterogeneity problems that can appear during discovery, composition, selection or invocation of Web services.

Process Mediator. The Process Mediator reconciles the public *process* heterogeneity that can occur during the invocation of Web services. It attempts to match the public processes of the client (represented by a Goal) with that of the service.

Communication Manager. The Communication Manager is responsible for dealing with the protocols for sending and receiving messages to and from DIP.

Resource Manager (Storage Manager). The Resource Manager manages the persistent storage for the DIP architecture. Persistence at the system level is required for all WSMO and nonWSMO entities used during operation of the DIP system.

Choreography Engine. A WSMO choreography defines how to interact with a Web service in terms of exchanging messages. The Choreography engine is used to enact the choreographies of the service requester (described in the Goal) and provider (described in the service).

WSML Reasoner. The WSML Reasoner is required by several other components in the architecture, notably discovery and both the process and data mediator components, to reason over the formal description of different aspects of the Semantic Web service description.

4.2 Goal Based Web Service Discovery

The following entry-point initiates this system behaviour.

getWebServices (WSMLDocument): Web services

A service requestor that requires a list of available Semantic Web services fulfilling its requirements can use this execution semantics by providing its requirements as a Goal description. A set of Web services matching the Goal is returned.

First, the Goal is received by the Communication Manager. Next, Discovery carries out matchmaking basing on the Goal contents. Data Mediation can be requested (*need DM*) during the Discovery when the ontologies of the Goal and SWS being matched differ. This Data Mediation may succeed, after which the Discovery can continue. The Data Mediation may also fail, after which a new WS is needed (the WS being checked cannot be mediated, and is useless for the given Goal).

The Discovery process continues (with or without Data Mediation) until a list of matching Web services is compiled. The list of Web services is complete when a preconfigured maximum number has been reached or there are no more known Web services to be matched against the Goal. The list of discovered Web services (*WS[]*) is returned as a result of the Discovery component. The list is empty if no Web services were found (*empty[]*). This means that none of the Web services known to the system could be matched to the service requester's Goal.

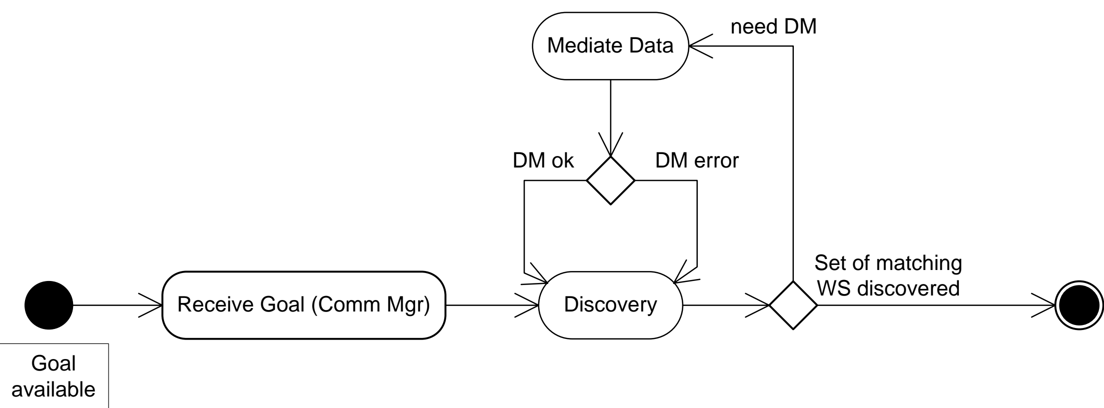

Figure 1: Goal centric Web Service Discovery in DIP

This behaviour is relevant when a decision about which Semantic Web service to execute is to be made outside the DIP system. The decision could be taken manually or by a Semantic Web service selection application based on the service requester's own service selection policies. The process of generating a list of Semantic Web services that meet a given Goal is shown in Figure 1. The Discovery service runs in a loop until the upper limit of Semantic Web service descriptions is reached or until there are no more descriptions to discover. The set of all discovered Semantic Web services is returned to the service requestor. It is then up to the requester to choose and to register communication with a specific Semantic Web service.

4.3 Goal Based Service Execution

The following entry-point initiates this system behavior:

achieveGoal (WSMLDocument): Context

A service requestor wishing to use DIP for all aspects of goal-based service invocation (discovery, mediation, invocation) provides both the goal and input data descriptions in a single WSML document. The returned Context is a unique identifier used to identify the instance of a conversation between the service requester and provider.

This scenario is based on the assumption that the service requestor is able to provide, up-front, all data required by the discovered Web service. Figure 2 depicts this execution semantic. First, discovery is carried out in the manner described in section 4.2.

After Discovery the Selection component selects the Web service from the list that best fits the user's preferences (however they are expressed).

Figure 1: Capability based service execution

Before the choreography for the service can be executed, it must be ensured that both requester and provider are able to understand each other's data. If necessary data mediation is used to mediate between data instances belonging to the ontology used by the Goal and that of the Web service.

The conversation between the service requester and the Web service then takes place according to their choreography descriptions. Process mediation may be also involved if the Choreography Engine cannot directly match the message exchange pattern published by the Goal and the Web service respectively. For example, it may change the sequence of messages, generate some dummy messages, preserve some messages to send later on or drop some received messages where appropriate.

Where data mediation and process mediation have completed (or were not needed) the Communication Manager is used to send the message to the discovered Web service.

4.4 The invocation of Web Services

The following entry-point initiates this system behavior.

invokeWebService(WSMLDocument, Context): Context

Once a service requestor knows which Semantic Web service is required to be invoked, a conversation has to be carried out with the DIP to provide all the necessary data to make the execution of this Semantic Web service feasible.

The Context (identifying a registered Web service to be invoked – see section 4.5) and the input ontology instance(s) are provided to DIP via the Communication Manger. In this version of the deliverable, we assume that any ontologies and a Goal description is included in the WSMLDocument parameter to enable the DIP system to be able to send messages back to the service requester. The conversation between the service requester and provider takes place according to the Goal and Web service choreographies respectively. The Context uniquely identifies the conversation and must be preserved by both interacting parties. This approach facilitates keeping track of the ongoing conversation status and refers to additional data required to make the communication between the two parties feasible (e.g. data related to Process Mediation such as previously received messages or status of internally initialized choreographies).

After this, the Process Mediation component is accessed to bridge (where possible) any differences between the choreography descriptions of the service requester and provider. For instance, it may change the sequence of messages, generate some dummy messages, and preserve some messages to send them later on or drop some received messages. In general, its goal is to enable communication between parties despite any differences in their communication patterns (choreographies).

If necessary, data mediation is also requested, when differences between used ontologies occur, to mediate between the ontology of the service requester and that of the provider. The ready message and recipient address is forwarded to the Communication Manager. After this, a message is sent asynchronously. Response message is received by the Communication Manager and then it is again passed on to the Process Mediation service. The process of invocation of Web Services is shown in Figure 3.

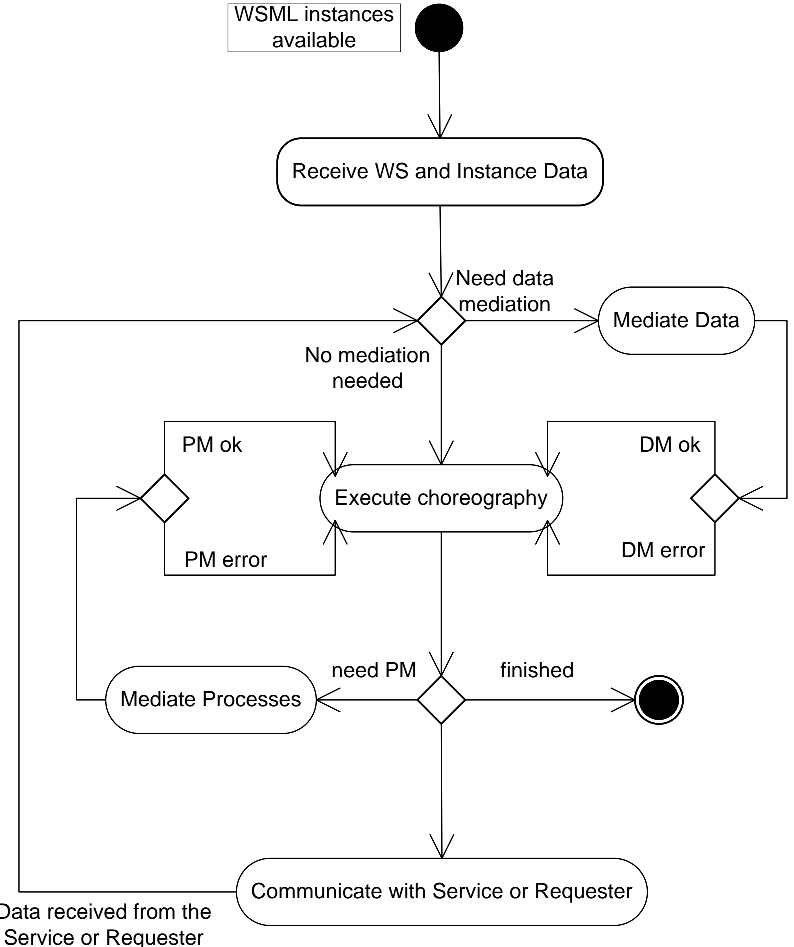

Figure 2: The invocation of Web Services

It should be stressed that the approach which presented in this system behavior allows seamless communication with the outside world. One type of Invoker and Receiver service can be exploited for communication with both interacting parties. From the point of view of these platform services, there is no differentiation in interacting with service requestor or Semantic Web service.

4.5 Register Execution with DIP Architecture

The following entry-point initiates this execution semantic:

registerCommunication(WebService):Context

This entry-point in invoked by requester in order to register the conversation with a chosen Semantic Web service. A unique Context identifier is assigned and requester and provider choreographies are internally initialized. This Context allows one to keep track of ongoing conversation between registered parties and makes the mediation feasible.

5 THE DIP ORCHESTRATION ONTOLOGY AND EXECUTION SEMANTICS

The technologies considered, as we discussed above, for describing execution semantics are UML, ASM and Cashew. The orchestration ontology developed in the DIP

deliverable 3.8 [14] is based on ontologized ASM which is capable of describing dynamics of Web service interface definitions [10]. Thus, it is possible to easily describe execution semantics using this orchestration ontology. In addition, this ontology is rich enough for describing execution-time requirements of orchestration where run-time binding between Web service components is necessary.

Execution semantics can be seen as an orchestration of the middleware services that go to make up the DIP architecture. They define the execution order of the components of an execution environment. At the same time, orchestration in DIP defines the means of realizing the functionality of a Web service by composition of other Web services (or Goals). Hence, the DIP orchestration ontology defined in deliverable 3.8 [14] provides an ideal candidate for describing the execution semantics and will be used in the final revision of this deliverable.

The orchestration ontology of deliverable 3.8 supports concurrency features which are one of the requirements of the execution semantics. It describes all resource descriptions and data elements interchanged between system entities based on ontologies. This indicates that the data elements that are exchanged between system components as demanded by execution semantics can be described using DIP orchestration ontology.

6 Summary

In this deliverable we described the standardisation activities with OASIS around execution semantics for semantics-enabled systems. We introduced the three layer approach to be taken to describe execution semantics based on the outcome of work package 3 on an ontology for choreography and orchestration. Four mandatory execution semantics for the DIP architecture were defined using UML activity diagrams. This is the first deliverable on this work since its alignment with the OASIS SEE TC deliverable, and, consequently, there is significant scope to refine the work, both in terms of the detail of the execution semantics presented and their formal descriptions. We expect to achieve this for the final version of this deliverable at the end of month 36 of the DIP project.

7 References

[1] Börger, E. (1999): *High level system design and analysis using abstract state machines*. In Current Trends in Applied Formal Methods (FM-Trends 98), number 1641 in LNCS, pages 1–43. 1999.

[2] de Bruijn, J.; Lausen, H.; Krummenacher, R.; Polleres, A.; Predoiu, L.; Fensel, D. (2005): The WSML Family of Representation Languages, WSML Working Draft v0.2, 2005, http://www.wsmo.org/TR/d16/d16.1/v0.2/

[3] Dijkstra, E.W. (1972): Notes on structured programming. *Structured Programming*, 1972.

[4] Eshuis, H. (2002): *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, Twente, Netherlands, 2002.

[5] Eshuis, R.; Wieringa, R. (2002): *Comparing Petri Nets and Activity Diagram Variants for Workfow Modelling - A Quest for Reactive Petri Nets*. In H. Ehrig, W. Reisig, and G. Rozenberg, editors, *Petri Net Technologies for Communication Based Systems*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2002.

[6] Harel, D.; Rumpe, B. (2004): *Meaningful Modeling: What's the Semantics of "Semantics"?*, *Computer*, vol. 37, no. 10, pp. 64-72, October, 2004.

[7] Norton, B. (2005): *Experiences with OWL-S, Directions for Service Composition*: *The Cashew Position* (2005). In OWLED 2005: OWL Experiences and Directions, 11-12 November 2005, Galway, Ireland.

[8] van der Aalst, W; ter Hofstede, A.; Baros, P. (2003): *Workflow Patterns*. Journal of Parallel and Distributed Systems 14(1) pp. 5–51, 2003.

[9] Zaremba, M.; Moran, M.; Haselwanter, T.; Sapkota, B. (2006): *Semantic Web Services Architecture and Information Model*, OASIS Working Draft v0-r8, 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex

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

[11] Norton, B., Foster, A. and Hughes, S. *A Compositional Operational Semantics for OWL-S*, presented at 2nd International Workshop on Web Services and Formal Methods (WS-FM), Versailles, France, 2005.

[12] Jensen, K., *Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Volume 1*, Basic Concepts of Monographs in Theoretical Computer Science. Springer-Verlag, 1994

[13] W. v. d. Aalst and A.H.M. ter Hofstede, "YAWL:Yet Another Workflow Language," presented at Accepted for publication in Information Systems, and also available as QUT Technical report, FIT-TR-2003-04, Queensland University of Technology,Brisbane., 2003.

[14] J. Lemcke, B. Norton, and C. Pedrinaci, "DIP WP3: Service Ontologies and Service Descriptions, D3.8 Ontology for Web Services Choreography and Orchestration," 2006.

[15] M. Moran, M. Zaremba, A. Mocan, E. Cimpian, T. Haselwanter, and M. Zaremba, "DIP Deliverable 6.11, Semantic Web Services Architecture and Information Model," 2006, available online at http://dip.semanticweb.org/deliverables.html