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

Deliverable

WP 6: Architecture & Interoperability
D6.16
Execution Semantics for Semantics-Enabled Systems,
Standardisation activity with OASIS

Matthew Moran
Barry Norton
Maciej Zaremba
Brahmananda Sapkota
Omair Shafiq
John Domingue

December 31st, 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.
Every software system has an execution semantics that provides a definition of how the system operates. In other words, it can be considered as the invocation order of the various components that go to make up the system. Usually a service-oriented system has one operational semantics defined at design-time. Aspects of the semantics may be configured through the use of configuration files or scripts but the essential flow of control and data remain fixed. In this deliverable, we present an approach to enabling multiple execution semantics over the same set of services in the DIP service-oriented architecture. This deliverable is the basis for the ongoing standardization work of the OASIS Semantic Execution Environment Technical Committee (OASIS SEE TC) for the definition of execution scenarios for Semantic Execution Environments for Semantic Web services.

**Keywords**

Execution semantics, behaviour, architecture
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Name(s)</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-Jun-06</td>
<td>002</td>
<td>Matthew Moran</td>
<td>Committee sent for review.</td>
</tr>
<tr>
<td>23-Oct-06</td>
<td>003</td>
<td>Barry Norton, Matthew Moran</td>
<td>Synchronise deliverable with the OASIS TC work</td>
</tr>
<tr>
<td>31-Dec-06</td>
<td>004</td>
<td>Matthew Moran Brahmananda Sapkota</td>
<td>Final synchronisation with OASIS TC deliverable. (OASIS SEE work carries on beyond this point).</td>
</tr>
</tbody>
</table>

**Reviewer**

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michal Zaremba</td>
<td><a href="mailto:michal.zaremba@deri.org">michal.zaremba@deri.org</a></td>
<td></td>
</tr>
<tr>
<td>Partner UIBK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laurentiu Vasiliu</td>
<td><a href="mailto:Laurentiu.vasiliu@deri.org">Laurentiu.vasiliu@deri.org</a></td>
<td></td>
</tr>
<tr>
<td>Partner NUIG</td>
<td></td>
<td></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 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: mmntnez@bankinter.es  
Tel: 916234238 |
| British Telecommunications Plc. | BT | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IPS 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  
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: mary@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 5070485 |
<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| 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 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Name</th>
<th>Address</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov</td>
<td>Ontotext Lab, - Sirma AI EAD</td>
<td><a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td>+359 2 9768 303</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Office Express IT Centre, 3rd Floor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>135 Tzarigradsko Chausse</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sofia 1784, Bulgaria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td>Tel.: +359 2 9768 303</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
<td>Unicorn Solutions Ltd,</td>
<td><a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td>+972 2 6491111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Malcha Technology Park 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jerusalem 96951, Israel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td>Tel.: +972 2 6491111</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer</td>
<td>Starlab- VUB</td>
<td><a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
<td>+32 (0) 2 629 3749</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vrije Universiteit Brussel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pleinlaan 2, G-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1050 Brussel ,Belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
<td>Tel.: +32 (0) 2 629 3749</td>
<td></td>
</tr>
</tbody>
</table>
**LIST OF KEY WORDS/ABBREVIATIONS**

**Keywords**
Execution semantics, Process, Dynamic model, State Machine

**Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Abstract State Machines</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>CASHEW</td>
<td>Composition And Semantic enHancement of Web Services</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>SEE</td>
<td>Semantic Execution Environment</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Service</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Discovery Description and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Service Modeling Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSMX</td>
<td>Web Service Execution Environment</td>
</tr>
</tbody>
</table>
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 ............................................................................................... 5
   3.3 Abstract State Machines – WSML ......................................................................... 5
4 OVERALL APPROACH .................................................................................................. ERROR! BOOKMARK NOT DEFINED.
   4.1 Overview of the DIP Architecture .......................................................................... 7
   4.2 Goal Based Web Service Discovery ....................................................................... 8
   4.3 Goal Based Service Execution ............................................................................. 10
   4.4 The invocation of Web Services .......................................................................... 12
   4.5 Register Execution with DIP Architecture ............................................................ ERROR! Bookmark not defined.
5 THE DIP ORCHESTRATION ONTOLOGY AND EXECUTION SEMANTICS ..................... 14
6 SUMMARY .................................................................................................................... 14
7 REFERENCES ............................................................................................................... 14

LIST OF FIGURES
   Figure 1: Capability based service execution ................................................................. 11
   Figure 2: Goal centric Web Service Discovery in DIP ..................................................... 9
   Figure 3: The invocation of Web Services .................................................................... 13
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. It 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 comprehensible manner thus appropriate modeling method has to be chosen.

UML Activity Diagrams depict a coordinated set of activities that capture the behavior of a modeled system. They are used to specify control and data flow between the entities providing language constructs that enable to model elaborate cases like parallel execution of entities or flow synchronization. For the DIP architecture, they are used to identify the platform services that are used in each specific execution semantics along with the control and data flow between them. In each of the activity diagrams provided in this document, the actions that provide steps in the execution semantics are identified inside a red dashed-line box. The platform services, for discovery, mediation, selection etc., are drawn as UML activities with interfaces that accept inputs and provide output as data objects. These interfaces are consistent with the defined DIP API.

In each of the provided activity diagrams, rounded rectangles with small square input pins denote actions. The pins represent input and output data ports. Rounded rectangles with large rectangular boxes at their boundaries (e.g. Communication Manager in figure 1) indicate activities that correspond to DIP platform services. The rectangular boxes at the boundaries represent parameters that the activity accepts as input or provides as output. Data flow directional arrows can be identified as those beginning or ending at a data pin or a parameter box. The actions inside the red boundary are the responsibility of the execution semantics. Control flow directional arrows have no associated data and go from action to action inside the execution semantics. Finally, the vertical thick black bars are data-flow branches. Data flowing into a branch is made available at all
outgoing flows. Where multiple input pins are available on an action, the action does not commence until data has arrived at each of those pins. Data becomes available on all outgoing pins of an action, as soon as that action completes.

A prototype reference implementation of the execution semantics is provided in the WSMX open-source application. In the prototype, each execution semantic is coded as a java class that maintains its own state persistence. One of the aims of OASIS SEE TC is to build on the work provided by the DIP orchestration ontology to enable a description-based execution semantics that can be independent of a particular programming language.

Due to the wide proliferation of UML notation several efforts were carried out to make concrete 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 Defined Execution Scenarios

We define execution semantics for three initial execution semantics identified for the DIP architecture and describe the details of each in the following subsections. The DIP architecture deliverable, D6.15 [15], provides a description of the service-oriented architecture designed for DIP. This architecture defines a set of platform services that are required to achieve the functionality of a Semantic Web services execution environment. An interface is defined for each platform services, defined in terms of the conceptual model provided by WSMO. The combination of all interfaces provides the basis for the DIP application programming interface (DIP API) which is intended to provide a programmatic platform against which follow-on projects to DIP can re-use the research and software developed over DIP’s lifetime.

Strongly helping this continuity is the fact that the DIP API has been adopted as the starting point for the OASIS Semantic Execution Environment API (OASIS SEE API). The intent is that, through OASIS, the DIP architecture and API becomes the industry-recognised recommendation for Semantic service oriented architectures.

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.

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 whose capabilities match that described in the WSMO Goal specified by the service requester. Where a service requester does not support the generation of WSMO Goals, a suitable adapter is required. In the DIP architecture, it is expected that the internal design of the discovery platform service requires both data mediation and a WSML reasoner.

**Selection.** The Non-functional Selector is used to select the most suitable service from a set of matching services identified during service discovery. For example, if the discovery results in more than one service that satisfies a Goal, this component may be used to choose one that should be invoked. How such a component should be designed is outside the scope of this document.

**Data Mediation.** Data Mediation is the task of reconciling the data heterogeneity problems that can appear during discovery, composition, selection or invocation of Web services. Data heterogeneity means that the data model, defined ontologically for the Web service, does not match the data model defined for the Goal.

**Process Mediation.** Process Mediation reconciles the public behaviour heterogeneity that can occur between the Goal and the Web service descriptions. Both the owner of the Goal and the provider of the Web service will have a defined message exchange pattern at the interface between their private implementations and how they wish to interact with the rest of the world. Process mediation uses these descriptions to map between the respective message exchange patterns, if possible. In the DIP architecture, the internal design of the process mediation platform service uses data mediation and a WSML reasoner.

**Communication Manager.** The Communication Manager is responsible for handling messages that are received by or sent from the DIP architecture. This platform service sits logically just inside the boundary of the DIP architecture and the outside world. Some of the responsibilities it has include, dealing with the protocols for sending and receiving messages, storage of intact messages immediately after receipt and immediately before sending by DIP, and the initiation of the execution semantics depending on the entry-point at which an incoming message has been received.

**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 non-WSMO entities used during operation of the DIP system. The
architecture defines the API for this platform service and one possible implementation is provided by the WSMO Studio [16].

**Choreography Engine.** A WSMO Web service choreography defines the public interaction offered by the Web service in terms of message exchange. The choreography of a WSMO Goal description defines the message exchange behaviour a service requester would like to find. The Choreography engine is used to enact the choreographies of the service requester (described in the Goal) and provider (described in the service). This means evaluating the state-machine descriptions of the respective choreographies against the knowledge basis of information provided as input along with any inferred information acquired via a suitable reasoning engine.

**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.

In each of the subsequent sections, execution semantics are linked to defined entry points to the DIP architecture. Entry points can be considered as input ports, to which messages can be sent, for initiating specific execution semantics. 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.

\[
\text{execSemName (Input data types): Return data type}
\]

### 4.2 Goal Based Web Service Discovery

The following entry-point initiates this system behaviour.

**getWebServices (WSMLDocument): Web services**

A service requestor wishing to discover a list of Semantic Web services fulfilling its requirements provides its requirements as a Goal description using WSML. A set of WSML Web service descriptions whose capability matches the Goal is returned.

Figure 1 shows the activity diagram. A message containing a WSML Goal is received and passed to the Communication Manager which takes care of whatever message persistence, unpacking and decryption that may be required. The extracted WSML Goal is passed to the Parser for transformation into the internal WSMO4j data representation used by DIP. The WSMO4j Goal is sent to the Discovery activity which looks for WSMO Web service descriptions with capabilities that match that of the Goal. The discovery activity may need data mediation as part of its internal design but this internal aspect of that activity is not modelled as part of this execution semantics. Discovery continues until a set of matching Web services is compiled. Assuming this set is non-empty, the selection activity determines which of the discovered services to return to the service requester. The WSMO4j description of the selected service must be transformed to WSML and packaged into a message that can be sent back to the requester. The Communication Manager takes care of this activity ending the execution semantics for Goal-based discovery.
Figure 1: Goal-Based Web service Discovery
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. A unique identifier for the conversation initiated by the receipt of the Goal is returned by the execution semantics to the service requester. In parallel the Goal and the WSML instance input data is processed by the execution semantics to discover and invoke a service whose capability matches that described by the Goal description.

This scenario illustrated in the UML activity diagram of figure 2 is based on the assumption that the service requestor is able to provide, up-front, all input data required by the discovered Web service.

The execution semantic is initiated by the receipt of the Goal and input data messages. As in figure 1, the Communication Manager activity is used to unpack the WSML content of the messages. Once unpacked, the WSML document is sent to the Parser activity where it is transformed into WSMO4j data objects. The output of the Parser is split. The Goal is sent to the Discovery and subsequent Selection activity to determine a matching Web service to invoke. The Goal is also sent to the Choreography activity which waits on the arrival of the other required inputs. The input data instances are sent to the Process Mediation activity, which waits for Discovery and Selection to complete. Once Discovery and Selection complete, the resulting WSMO4j Web service is sent to both the Choreograph activity (which still waits) and the Process Mediation activity.

Process Mediation now has all input data at its input pins and can commence. We do not describe the internal design of the Process Mediation activity here but we do note that it is responsible for matching the message exchange patterns required by the Goal and offered by the Web service description. This matching includes both the message patterns and the data types used by the respective messages. Hence, data mediation may be required as an internal operation. We do not model this as part of the execution semantics. Process mediation determines how the input data instances need to be organised so that they can be understood by the Web service choreography description. The output of Process Mediation is WSMO4j data instances organised in this manner. Once, these are available, the Choreography activity has all required input and (most likely using a reasoning engine as part of its internal design) determines which messages need to be sent out based on the choreography description (an abstract state machine) of the Web service. This WSMO4j data must be transformed into a format defined by the Web service grounding information (taken from the WSDL service binding) and sent to the Web service endpoint. This activity is carried out by the Communication Manager.

Note that this execution semantic, without the discovery and selection activities, could be used to handle messages returned from the Web service to be sent back to the service requester, assuming that the service requester provided an invocable endpoint at which it could receive such messages.
Figure 2: Goal-Based Service Execution
4.4 Web Services Invocation

The following entry-point initiates this system behavior.

invokeWebService(WSMLDocument, Context): Context

Where the service requester already knows which Semantic Web service they wish to invoke, the execution semantics illustrated in Figure 3 is used. The first input signal provides the WSML content while the second provides a conversation context. If this is the first message being sent to the Web service, there will be no existing conversation and this context signal will be empty. As in the other two scenarios, the WSML input message is unpacked by the Communication Manager activity and transformed into WSMO4j by the Parser activity. In this case, however, the WSML input contains both the target Web service description and the input instance data. The (possibly empty) context is sent to an input pin of the Choreography activity which waits for the other inputs to arrive. After the WSML has been parsed, the WSMO4j Web service description and the instance data are passed to the Data Mediation activity to handle any data mapping that may be required (may require a WSML reasoner to be invoked internally). After the Data Mediation, all required inputs are available to the Choreography activity. It determines what message(s) should now be sent to the target Web service and provides this instance data as output, along with grounding information to the Communication Manager activity. This packages the actual message that must be sent to the target Web service endpoint and includes a conversation context, extracted from the grounding data. The context is then available to the Web service receiving the message so that it can be used in any future messages that it may wish to send back to the service requester.
Figure 3: The invocation of Web Services
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


