DIP
Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP6: Architecture and Interoperability

D6.11

DIP Semantic Web Services Architecture and Information Model

Matthew Moran, Michal Zaremba
Adrian Mocan, Emilia Cimpian
Thomas Haselwanter, Maciej Zaremba

June 29, 2006
SUMMARY

This deliverable provides a description for the DIP Semantic Web services architecture following the guidelines of the IEEE Recommended Practice for Architectural Descriptions of Software-intensive systems (IEEE 1471) [5]. This standard proposes the description of an architecture identifying concerns that should be addressed, stakeholders in the system and viewpoints describing different perspectives on the architecture. Four viewpoints are presented in this description. Within the DIP consortium, the intended audience for this deliverable are the members of the technical work packages (WP1 – WP5), as well as the project’s technical committee. Outside DIP, we hope this deliverable is useful for IT professionals interested in the architectural description of an execution environment for Semantic Web services including the components the architecture contains and how they relate to each other.

A high-level overview relates the architecture to the DIP Description of Work (DIP Technical Annex), describes the relationship with the Web Service Modeling Ontology family of specifications (WSMO/L/X), and introduces the underlying principles of the architecture. The Peer-to-Peer (P2P) viewpoint looks at how the architecture could be incorporated in a distributed system of cooperating peers where each peer is an implementation of the architecture. The structural viewpoint identifies the functional components of the architecture and their interfaces. Finally, the behavioural viewpoint looks at the event-based messaging design for the DIP architecture and explains how multiple concurrent execution semantics for the architecture may be put in place.

The architecture presents a vision of a potentially distributed architecture, where semantically described components can be plugged in and out, and where the description of the invocation order of the components is separated from their implementation. While we assume that for the Semantic Web services (SWS) infrastructure, a minimal set of components for discovery, mediation, composition, invocation etc. are required, the system should support the hosting of additional components, whose functionality may be unknown at system design-time. The architecture is designed to allow new or upgraded components to be plugged in and out without necessarily requiring a rebuild or restart the system.

The prototype for the DIP architecture, WSMX is provided as open-source software at SourceForge.net and formed an integral part of the W3C member submission for Semantic Web services submitted by DERI, Galway (NUIG) and DERI Innsbruck (Leopold Franzens Universitaet, Innsbruck). The DIP architecture and associated execution semantics also provide the basis for the newly formed OASIS Technical Committee for specifying a Semantic Execution Environment (OASIS SEE TC). The focus of this OASIS committee is to further develop the Semantic Web services architecture and description of execution semantics initiated through WSMX.

The architecture described here is relevant for all deliverables in DIP responsible for providing a functional component for the DIP Semantic Web services execution environment prototype.

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.
This deliverable describes the Architecture for the DIP project. The document provides both a high-level overview of the necessary system components (structural view) and the dependencies and interactions between them (behavioural view). The deliverable provides a blueprint for the development of the DIP system prototype implemented in the Web Service Execution Environment (WSMX).
<table>
<thead>
<tr>
<th>Reviewers</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandra Stincic</td>
<td><a href="mailto:sandra.stincic@bt.com">sandra.stincic@bt.com</a></td>
</tr>
<tr>
<td>Partner BT</td>
<td></td>
</tr>
<tr>
<td>Bernhard Schreder</td>
<td><a href="mailto:bernhard.schreder@hanival.net">bernhard.schreder@hanival.net</a></td>
</tr>
<tr>
<td>Partner Hanival</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 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 |
| British Telecommunications Plc.              | BT      | Dr. John Davies
BT (Orion Floor 5 pp12)
Adastral Park Martlesham
Ipswich IP5 3RE,
United Kingdom
Email: john.nj.davies@bt.com
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL    | Prof. Karl Aberer
Distributed Information Systems Laboratory
École Polytechnique Fédérale de Lausanne
Bât. PSE-A
1015 Lausanne, Switzerland
E-mail: Karl.Aberer@epfl.ch
Tel: +41 21 693 4679 |
| Essex County Council                         | Essex   | Mary Rowlatt,
Essex County Council,
PO Box 11, County Hall, Duke Street,
Chelmsford, Essex, CM1 1LX,
United Kingdom.
E-mail: maryr@essexcc.gov.uk
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik                 | FZI     | Andreas Abecker
Forschungszentrum Informatik
Haid-und-Neu Strasse 10-14
76131 Karlsruhe
Germany
E-mail: abecker@fzi.de
Tel: +49 721 96540 |
| Institut fur 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 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Address</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td>Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France</td>
<td>E-mail: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a> Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany</td>
<td>E-mail: <a href="mailto:ts@inubit.com">ts@inubit.com</a> Tel: +49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain</td>
<td>E-mail: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a> Tel. +34 913 349 797</td>
</tr>
<tr>
<td>Hanival WEB Solutions</td>
<td>Alexander Wahler Hanival Internet Services GmbH Niederacher &amp; Wahler OEG, Kirchengasse 13/1a A-1070 Wien E-mail: <a href="mailto:wahler@hanival.net">wahler@hanival.net</a> Tel. +34 913 95843 11</td>
<td></td>
</tr>
<tr>
<td>MDR Partners</td>
<td>Rob Davies MDR Partners 8 St. Andrew Street &amp; Hertford, Herts, United Kingdom, SG14 1JA, E-mail: <a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a> Tel.: +44 (0)208 8763121</td>
<td></td>
</tr>
<tr>
<td>The Open University</td>
<td>Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a> Tel.: +44 1908 655014</td>
<td></td>
</tr>
<tr>
<td>SAP AG</td>
<td>Dr. Elmar Dorner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a> Tel: +49 721 6902 31</td>
<td></td>
</tr>
<tr>
<td>Sirma Al Ltd.</td>
<td>Atanas Kiryakov, Ontotext Lab, - Sirma Al EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria E-mail: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a> Tel.: +359 2 9768 303</td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>Contact Information</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg&lt;br&gt;Unicorn Solutions Ltd,&lt;br&gt;Malcha Technology Park 1&lt;br&gt;Jerusalem 96951,&lt;br&gt;Israel&lt;br&gt;E-mail: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a>&lt;br&gt;Tel.: +972 2 6491111</td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer,&lt;br&gt;Starlab- VUB&lt;br&gt;Vrije Universiteit Brussel&lt;br&gt;Pleinlaan 2, G-10&lt;br&gt;1050 Brussel, Belgium&lt;br&gt;E-mail: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a>&lt;br&gt;Tel.: +32 (0) 2 629 3749</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

1 **Introduction**  
   1.1 Organization of the Document .......................... 2  
   1.2 Change History ........................................ 2  
   1.3 Acknowledgement ....................................... 3

2 **Architecture Overview Viewpoint**  
   2.1 Addressing Requirements ............................... 4  
      2.1.1 Provision of an Open-Source Realisation of a SWS Architecture 4  
      2.1.2 Standardisation .................................. 5  
   2.2 Motivation for the Architecture ....................... 5  
   2.3 Two Fundamental Principle of the Architecture ........ 6  
   2.4 Relationship to WSMO and WSMX ....................... 6

3 **Viewpoint on the Architecture in the context of a Peer to Peer Network**  
   3.1 A P2P Network of DIP Nodes ........................... 8

4 **Structural Viewpoint**  
   4.1 Overview ............................................. 10  
   4.2 Core Component ...................................... 13  
   4.3 Resource Manager .................................... 13  
   4.4 Service Discovery .................................... 13  
   4.5 Non-functional Selector .............................. 14  
   4.6 Data Mediator ....................................... 14  
   4.7 Process Mediator ..................................... 14  
   4.8 Communication Manager ................................ 15  
   4.9 Choreography Engine ................................. 15  
   4.10 Parser ............................................... 16  
   4.11 Orchestration ....................................... 16  
   4.12 WSMLReasoner ....................................... 16  
   4.13 External Systems .................................... 17  
      4.13.1 WSMO Studio .................................. 17  
      4.13.2 Adapter Framework .............................. 17  
   4.14 DIP API ............................................ 17

5 **Behavioural View**  
   5.1 Informal Overview of Execution Semantics and Messaging in DIP . . 19  
   5.2 One-way Goal Execution Scenario ........................ 21  
   5.3 Conversation-based Goal Execution Scenario ............ 22  
   5.4 SWS Challenge ....................................... 22

6 **Summary** .............................................. 24
1 INTRODUCTION

This deliverable describes the DIP architecture for a Semantic Web service execution environment. The DIP work packages 1-5 provide the basic conceptual and technical building blocks required to set up DIP's infrastructure while this deliverable describes how these building blocks are consolidated into a coherent software architecture that can be used as a blueprint for implementation.

Both the terms Web service and Semantic Web service are used throughout this document. The intent is that Web service corresponds to a service described using WSDL, available at a URI, that can be invoked on the Web using a combination of SOAP, HTTP and XML. Semantic Web services are Web services for which a semantic annotation has been created allowing for richer possibilities for discovery, composition and mediation of Web services.

The IEEE Recommended Practice for Architectural Descriptions of Software-intensive systems (IEEE 1471) [5] recommends that software architecture descriptions be organized into one or more constituents called architectural views. Each view is used to represent the concerns of a particular viewpoint of one or more stakeholders in the architecture. In this context, stakeholders have an interest in or concerns relative to the the system. Concerns include system considerations such as performance, reliability, scalability, security etc.

A system stakeholder is defined as ”an individual, team or organization (or classes thereof) with interests in or concerns relative to a system”. The purpose of describing a system architecture is to record the architecture in such a way that it addresses the concerns and interest of these various perspectives. In this revision of the architecture we identify five categories of stakeholders and their primary interest in the architecture description provided by this deliverable.

- Owner of the system
- Acquirer of the system (e.g. a project considering the adoption of the DIP architecture as a starting point for their work)
- Designer or developer of a system based on the the architecture
- Component owner
- Case study partner

The concerns that we aim to address in the context of this architecture description are:

- Does the architecture achieve what is required of it as described in the DIP Project Description of Work (internal project document)?
- What is the relationship to the Web Services Modeling Ontology/Language/Execution Environment (WSMO/L/X)?
- What are the underlying principles for the architecture?
- How does the DIP architecture relate to peer to peer networks?
• Is there sufficient information and detail in the description to progress to a detailed technical design?

• How are individual technical components (e.g. prototypes from the technical work packages 1-5) plugged into the architecture?

• How can case study partners use the architecture to build their prototypes?

IEEE 1471 defines views as "a representation of a whole system from the perspective of a related set of concerns". Viewpoints then are described as a "pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis." We will define four viewpoints that correspond to the perspectives of each of the stakeholders. The following four viewpoints are described in this deliverable:

• an overview of the architecture including its relationship to WSMO and WSMX and how the architecture addresses the requirements in the DIP Project Description of Work

• the architecture in the context of a P2P network

• its structure in terms of functional components and their interfaces

• its behaviour in terms of interactions between components

1.1 Organization of the Document

The rest of this document is organized as follows. Section 2 looks at how the architecture addresses the requirements of the DIP Description of Work and introduces the architecture’s fundamental principles. Section 3 provides an overview of the architecture in the context of a P2P network. Section 4 provides the structure in terms of functional components and their interfaces. Section 5 describes its behaviour in terms of interactions between components. Section 6 provides a summary.

1.2 Change History

The first version of the architecture (M12) represented a high level approach identifying the main components required for a Semantic Web services architecture. As shown in figure 1.1, the second and the third versions of the architecture (M18 and M24) refined the architecture and focused on the design of the Core component and the integration of the functional component prototypes coming from work packages 1-5. In this version, we follow up on the recommendations from the second DIP review and align the architecture description with the recommendations of the IEEE 1471 Recommended Practice for the Architectural Description of Software Intensive Systems. In the context, we have introduced viewpoints on the architecture and have both revised earlier and created new UML models to help describe these viewpoints. The final version of the DIP Architecture (M36) will include further refinements of the model descriptions, going into further details where necessary, with the aim of providing a comprehensive architectural description of the DIP system which can be
used as a blueprint for further research in the field of Semantic Web service execution environments.

Figure 1.1: Revision History of the DIP Architecture

1.3 Acknowledgement

The work described in this deliverable was carried out jointly under the auspices of the DIP project funded by European Union’s IST programme (no. FP6 – 507483) and the DERI-Lión project funded by Science Foundation Ireland (SFI grant no. SFI/02/CE1/I131). The editors would like to thank the members of the WSMO, WSML, and WSMX working groups for their advice and input into this document.
2 Architecture Overview Viewpoint

This viewpoint is intended to motivate the DIP architecture in context and to describe how the architecture addresses the requirements set out in the DIP Project Description of Work (internal project document). The two fundamental principles underlying the architecture are introduced and the relationship of the architecture to WSMO, WSML and WSMX is explained.

This viewpoint is intended to address the following three concerns:

- Does the architecture achieve what is required of it in the DIP Project Description of Work?
- What is the relationship to WSMO/L/X?
- What are the underlying principles for the architecture?

All stakeholders listed in section 1 are addressed by this viewpoint. In particular the owner and acquirer stakeholders may be interested in the motivation and guiding principles behind the architecture as well as its relationship to the WSMO, WSML and WSMX working groups.

The rest of the chapter is organized as follows. Section 2.1 indicates how the architecture addresses the requirements set out in the DIP Project Description of Work. Section 2.2 describes the motivation for the architecture. Section 2.3 introduces two fundamental principles adopted by the architecture. Section 2.4 describes the relationship of the architecture to WSMO, WSML and WSMX.

2.1 Addressing Requirements

2.1.1 Provision of an Open-Source Realisation of a SWS Architecture

The requirements for the DIP architecture form a thread running through the entire DIP Project Description of Work. Chapter 1 states that "One of the key deliverables of DIP is an open source Semantic Web service architecture that is made public on a world-wide basis. The goal of the open source approach is to make the architecture easily available for other organizations (research as well as industrial). Through this approach a fast uptake of the work of DIP will be achieved". Again in section 4.1.1 it is stated that the "first main achievement of DIP will be the Semantic Web service architecture that will define a scalable platform for mechanized Web Service discovery, composition, and invocation. We will develop significant parts of this infrastructure in open source format in order to speed up the dissemination process of our efforts, ensuring that it will be widely if not globally used”.

The description of the DIP architecture, the definition of the DIP architecture API and the implementation of WSMX as an open-source prototype implementation of the DIP architecture go a long way to satisfying the requirements in this regard. The open-source nature of the DIP architecture prototype and its use of WSMO to define the domain models for all messages exchanged between components of the architectures have also already encouraged other EU FP6 projects such as COCOON e-Health and

\[1\]http://www.cocoon-health.com/
the Integrated Project, SUPER\(^2\), to pick up this work as a foundational input for their research.

2.1.2 Standardisation

The requirement for the DIP architecture to drive standardisation activities is raised in the DIP Project Description of Work: "the openness of eCommerce cannot be achieved without standardisation. This lesson can be learned from the success of the Web. But the standardisation requirements for eCommerce will be much more stringent. This will require standardisation of the actual content and business logics that are exchanged, which goes far beyond the requirement of standardising protocols and document layouts. DIP will develop significant contributions to this area (see also standardisation)" and again in section 5.1: "standardization of Web Services technology, e.g. W3C Web Services Activity, OASIS, WS-I. The project will closely monitor these standardization efforts, will provide reference implementations for some of the emerging standards in this area, and will participate in interoperability events”.

The related efforts of WSMO/WSML and WSMX as developed through DIP have been accepted as a member submission to the W3C activity on standardising Semantic Web services\(^3\). Additionally, at the beginning of November 2005, a new Technical Committee (TC) was formed at the OASIS eBusiness-standards consortium. The new committee is called the Semantic Execution Environment TC (SEE TC) and has the mission to develop guidelines, justifications, and implementation directions for deploying Semantic Web services in service-oriented architectures. The SEE TC came about as a direct result of dissemination activities carried out under the auspices of DIP. In April 2005, Michal Zaremba and Adrian Mocan (DERI, National University of Ireland, Galway) travelled to the OASIS Symposium on the Future of XML Vocabularies to present a tutorial on WSMO/WSML/WSMX. After the tutorial, they were approached by Patrick Gannon, director of OASIS and were invited to make a proposal for a Technical Committee at OASIS based on the work carried out in the WSMX open-source development of the DIP architecture.

2.2 Motivation for the Architecture

In general where advances in computer and information science occur their uptake is greatly assisted by having tools and frameworks that allow both software engineers and potential users to try out the new technology. The DIP architecture aims to address both these interest groups. It provides a blueprint for software engineers interested in designing a software system taking advantage of Semantic Web services.

For designers and users of Semantic Web services, the DIP architecture prototype allows the focus to be directed on learning how to maximize the benefits offered by being able to unambiguously describe the functional and non-functional aspects of Web services without having to invent a framework that knows how to handle those descriptions. The DIP architecture describes the glue that brings the features promised by Semantic Web service technology to life. In the same way as middleware systems hide the detail of how they link autonomous heterogeneous applications and systems

\(^{2}\text{http://super.semanticweb.org/}\)

\(^{3}\text{http://www.w3.org/Submission/2005/06/}\)
together, a Semantic Web services environment such as that of DIP hides the detail and complexity of how the requester and provider of Semantic Web services can communicate with each other even where they do not know each other in advance, speak different languages and expose different behaviours at their service interfaces.

2.3 Two Fundamental Principle of the Architecture

The DIP architecture adopts two principles described in the earlier work on the Web services Modeling Framework (WSMF) [3].

- **Component decoupling**
  This is a principle of SOAs that emphasizes the strong de-coupling of the various components that realize a software system. Making components self-contained supports a clear separation of concerns. Each component has a well defined functionality that can be utilized by other components.

- **Standardization of external behaviour of components**
  The external interfaces of the components within the DIP architecture are not expected to change very often. By having well defined component interfaces and behaviour descriptions, we separate the implementation of the individual components from the operation of the system as a whole. One of the long-term aims for the DIP architecture is the support of dynamic execution semantics. Execution semantics provide a formal description of execution scenarios for a system. They are an explicit description of the control and data flow of the system.

  By dynamic execution semantics, we mean that it should be possible to load an execution semantics description to an implementation of the DIP architecture at runtime. The system should be able to accept the description and be able to execute it without having to restart. For example, a particular deployment of The DIP architecture might never require data mediation because all data and processes available to the the DIP architecture are homogeneous. In such a case an execution semantics might be defined to ignore this component. This can be relevant in systems where, for optimization reasons, even a pass-through call to a component where the component takes no action might be too time consuming.

  We can also imagine scenarios where a deployment of the DIP architecture is provided with a new component whose functionality was not considered in the overall the DIP architecture design - it may be a specific requirement for that specific deployment. This could lead to a requirement to extend the principle of standardized component interfaces in the future.

2.4 Relationship to WSMO and WSMX

The Web Service Modeling Ontology (WSMO) [12] defines the conceptual model for Semantic Web services used within the DIP project. While WSMO provides the model, the Web Service Modeling Language (WSML) [1] provides a family of languages to express the elements of WSMO. Messages exchanges at the interfaces of the functional
components belonging to the DIP architecture are defined in terms of WSMO concepts expressed using WSML.

The Web Service Execution Environment (WSMX) [6] is a design and open-source implementation of infrastructure for executing Semantic Web services. Hand-in-hand with WSMO, WSMX provides a reference implementation of a software system with WSMO descriptions at its heart. Both the architectural style implemented by WSMX and the architectural style of applications executed through WSMX are that of a service oriented architecture (SOA).
3 Viewpoint on the Architecture in the Context of a Peer to Peer Network

Section 4 describes a structural view of the DIP architecture indicating its functional components and their public interfaces and may lead to the conclusion that the DIP architecture represents only a centralised middleware systems. However, this is not the case as an important aspect of the architecture is how it fits into the pattern of a distributed software system. Each implementation of the DIP architecture may be considered as a node in a peer to peer (P2P) network containing some or all of the functional components described in section 4. For example while many nodes may provide implementations of all functional components defined in the DIP architecture, another node may focus on providing a specialized functionality such as discovery with minimal implementation of the other components.

The first stakeholders interested in this viewpoint are designers/developer who are interested in using the DIP Semantic Web services architecture in a distributed P2P networks. Other stakeholders are the acquirer of implementation of the architecture. In both cases the concern being addressed is how the DIP architecture fits into a distributed software system.

3.1 A P2P Network of DIP Nodes

In general a P2P network is one in which there is no central controlling entity. Each node in the network can communicate with any other node to co-operate together in performing some task(s). The most well known P2P networks in recent years have been concerned with sharing video and audio files among a community of users. Each user has software enabling the P2P network on their machine and the addresses of machines in the network are available to all users of the P2P services. In reality, many networks advertised as P2P have a central node used for member management. Having a central node means having a central point of failure. With the DIP architecture we aim for a true P2P system where the failure of any node in the network will not result in the entire network breaking down.

P2P networks offer the advantage of allowing data and functionality to be split up across the network while at the same time allowing peers to co-operate in achieving tasks requiring this data or functionality even though it may be distributed across different locations. From the the perspective of the DIP architecture, a typical use-case is for service discovery. For example, a DIP node could receive a goal description and use its local discovery component to search for a match from the Semantic Web service descriptions known to the service registry at that node. Where no match is found, in a P2P network, the DIP node could forward the goal description to another DIP node known to it. Again if the second node has no success, it could forward the request again to a third DIP node and so on. In such a scenario, there would likely be a need for policies in place to control the depth of recursion allowed.

As each DIP node will be itself a Web service with a semantic description, the invocation of a component in another DIP node in the network will be the same as for any other Web service invocation. Figure 3.1 shows how a simple P2P DIP network with three nodes would look (with WSMX as the implementation of each DIP node). In the figure each DIP node has a different back end system associated
with it. For the discovery example mentioned earlier, each DIP node may have different service descriptions in its registry corresponding to specific functionality offered by the associated back end application.

A P2P network architecture of DIP nodes is a long term vision for the DIP architecture. It is likely that the development of this aspect of the DIP architecture will be driven by the requirements of service discovery as semantically annotated Web services are situated around the Internet and it is extremely unlikely that one single global registry of service descriptions will ever exist.
4 Structural Viewpoint

This section describes the structural viewpoint of the DIP architecture. It identifies the functional components defined for the current version of the architecture. It is likely that an industrial-strength design for a Semantic Web services architecture would require additional functional components such as those handling transactions, security and reliability concerns found in existing middleware software systems. The stakeholders for this viewpoint are software engineers and computer science researchers concerned with the functional components required to take full advantage of semantic annotations of Web service descriptions.

The section is organized as follows. Section 4.1 looks at the overall structural view of the DIP architecture showing all functional and data storage components for DIP, as well as supporting components that fall outside the scope of the DIP architecture itself (e.g., adapters, service-description editors). Subsequent sections describe the functionality of each component including a description of the component's interface and, where available, a reference to the respective components specification. Section 4.13 briefly explains two external systems that are important to, but not directly part of, the DIP architecture. Finally, section 4.14 points to the URL where the Java API for the DIP architecture is documented.

4.1 Overview

The DIP architecture consists of a set of loosely-coupled components as presented in Figure 4.1. It is a requirement that DIP functional components can be plugged in and unplugged without having to restart the system. The rationale for this requirement is that the architecture should allow for the inclusion or exclusion of new components at run-time as they are required. The description of the invocation order of components in implementations of the architecture are explicitly described in terms of execution semantics. The DIP architecture allows for multiple execution semantics to run on the same system concurrently. For example, one execution semantics could support only a complete end-to-end goal-based discovery, mediation and invocation of a Web service. Another execution semantics could be restricted to just the execution of data mediation.

The architecture diagram of Figure 4.1 shows an overview of the functional components in the DIP architecture in the context of two related external systems. The two components above DIP represents user-interface tools for visualizing and managing the four top-level elements of WSMO (goals, Web services, ontologies and mediators) and for creating mappings for data and process mediation. This can be provided by a WSMO compliant tool such as the WSMO Studio\(^1\) or the Web Service Modeling Toolkit (WSMT)\(^2\) [9]. The second external system is the Adapter Framework, shown between DIP and the service requesters and providers on the left of the Figure 4.1. DIP uses the conceptual model of WSMO to provide the data-type definitions used internally. In the architecture prototype this is realized using the WSMO4j object library\(^3\). Communication between DIP and the outside world is carried out using

\(^1\)http://www.wsmostudio.org/
\(^2\)http://sourceforge.net/projects/wsmt
\(^3\)http://wsmo4j.sourceforge.net/
Figure 4.1: DIP Architecture
WSML messages. Where an external system wishing to use DIP does not support WSML, an adapter is required to translate between the export format of the external system and DIP e.g. between a proprietary XML Schema and WSML. This translation is often referred to as lifting (from a less expressive to a more expressive language) and lowering (from a more expressive to a less expressive language).

Each of the following subsections briefly describes the functional components shown in Figure 4.1. Figure 4.2 shows a UML2 component model showing each component's interfaces and dependencies on other components.

Figure 4.2: UML2 Component Diagram for the DIP Architecture
4.2 Core Component

The Core Component takes the role of component co-ordinator for the DIP architecture. It manages interaction between the other components through the exchange of messages containing instances of WSMO concepts expressed in WSML. The Core component provides the microkernel and messaging infrastructure for the architecture.

As a side note, the architecture-prototype core is realized as a Java Management Extensions (JMX) microkernel as described by Haselwanter in [7]. The JMX technology is defined in three levels by the Java Community Process JSR003 as the instrumentation level, the agent level and the distributed services level.

The Core Component has three interfaces -

- **ManageSystem.** This is a management and monitoring interface to start and stop the system and allow its health to be monitored.
- **ManageExecutionSemantics.** Execution semantics specify the desired operational behaviour of the DIP architecture. Multiple definitions of execution semantics are supported and multiple instances of each execution semantics may run concurrently. This is described in section 5.
- **ManageMessaging.** The Core acts as a message bus which accepts messages and routes them to a suitable target component. Multiple concurrent message exchanges may be maintained.

4.3 Resource Manager

The ResourceManager is the component that manages the persistent storage for the DIP architecture. Persistence at the system level is required for all WSMO entities used during operation of the DIP system as well as the nonWSMO entities e.g. representing messages and their states. Individual functional components are responsible for any additional persistence that their design requires.

The DIP-architecture prototype uses WSMO4j, developed for DIP to provide an implementation of WSMO as a Java model.

There are two interfaces for this component -

- **WSMORepository.** Manage persistence for descriptions of WSMO descriptions of Web service, Goals, Ontologies and Mediators.
- **NonWSMORepository.** Manage persistence for entities required by the operation of the architecture e.g. messages, execution semantics descriptions etc.

4.4 Service Discovery

The Discovery component is concerned with finding Web service descriptions that match the goal specified by the service requester. A service requester provides the WSMO description of a goal they wish to achieve (described in terms of a desired capability with preconditions, assumptions, effects and postconditions). This is matched to the

---

4http://www.jcp.org/en/jsr/detail?id=3
5http://wsmo4j.sourceforge.net/
WSMO description of Web services known to DIP (described in terms of offered capabilities). The discovery component returns a (possibly empty) list of Web service descriptions. The specification for the Discovery component is available to the DIP consortium only in the deliverable D4.8 [4].

There is one interface called Discovery with two distinct operations - Discovery. On receipt of a WSMO Goal description, the component returns a set of matching services. QoSDiscovery. On receipt of a WSMO Goal description and a ranking ontology, the component carries out quality-of-service based discovery and returns a ranked list(s) of services with information on the quality-of-service concepts used for the ranking.

4.5 Non-functional Selector

The Non-functional Selector (labeled Selection Figure 4.1 is a component used to select the most suitable service from a list of matching services matched by discovery. For example, a service requester may define preference for the selection of the most suitable discovered Web service. If discovery results in more than one service that satisfies the goal, this component may be used to choose one based on specified preferences.

This component has a single interface - NonFunctionalSelector.

4.6 Data Mediator

The DIP DataMediator component has the role of reconciling the data heterogeneity problems that can appear during discovery, composition, selection or invocation of Web services. It is expected that this component should use inter-ontology mappings described in a suitable mapping language (defined at design-time) to implement the required mediation. Since the mappings will need to be stored one of the dependencies for the component is to the NonWSMOInterface of the Resource Manager. Accessing WSMO entities means a dependency on the WSMOInterface of the Resource Manager. The mappings required by the mediator will need to executed by a suitable logical reasoner. Hence there is a further dependency to the Reasoner component. The DIP Data Mediation module specification is available to the consortium only in the deliverable D5.2 [10]

The Data Mediator has one interface - DataMediator

4.7 Process Mediator

The DIP Process Mediator has the role of reconciling the public process heterogeneity that can occur during the invocation of Web services. That is, ensuring that the public processes of the client (represented by a Goal) and the matching service match. Since both the client and the Web service publish their public processes as choreographies, and the public processes are executed by sending/receiving messages, the
process mediation component will deal with reconciliation of message exchange patterns based on choreography. Where data mediation is necessary, it is expected that it takes advantage of the data mediation component. The DIP Process Mediation module specification is available to the consortium only in the deliverable D5.3a [2]

The Process Mediator has one interface - **ProcessMediator**

### 4.8 Communication Manager

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

There are four required interfaces -

**EntryPoint.** This interface provides the entry point for initiating an instance of an execution semantics in the DIP system. A client may send the DIP system a Goal they wish to achieve as well as the input data (ontology instances) corresponding to that Goal. The Goal and input data can be sent together or in separate messages. The Receiver expects the contents of all messages it receives to be expressed in WSML. If the client uses a different language, then an adapter mechanism is required. It is the EntryPoint interface of the CommunicationManager component that has responsibility for initiating an execution semantics corresponding to the type of received message.

**Invoker.** The Invoker interface is used whenever a Web service needs to be invoked by the DIP system. The invoker receives the WSMO description of the service, the endpoint for the invocation and the data that that particular endpoint expects to receive. It is responsible for making the actual invocation of an operation on a service. In the majority of existing Web service implementations, this means ensuring that the semantic description of both the data and the behaviour of the Web service are grounded to the corresponding WSDL descriptions. A grounding (semantic to syntactic) mechanism is required to translate between the semantic messages (in WSML) and the syntactic messages (in XML) defined for the majority of Web services using WSDL.

**Receiver.** The receiver interface accepts messages that correspond to conversations that have already been initiated between a client and a service via the DIP system. Either the client or the service sends a message to this interface including a context that indicates to the system the sending party and an identifier to the conversation.

**WSMORegistry.** The WSMORegistry interface exposes the methods for the DIP system to act as a repository of WSMO entities.

### 4.9 Choreography Engine

A WSMO Choreography defines how to interact with a Web service in terms of exchanging messages - the so-called communication patterns.

After discovering a Web service description, one has to know the observable behaviour of the Web Service in order to achieve the desired functionality. Choreography and Orchestration comprise part of the interface definition of a WSMO service description. Choreography describes how to communicate with the service such that the service will provide its capability. Orchestration describes how the service collaborates with
other WSMO services to achieve its capability. The **Choreography Engine** is responsible for using the choreography descriptions of both the service requester and provider to drive the conversation between them. It is the responsibility of the engine to maintain the state of a conversation and to take the correct action when that state is updated. For example, the update of the state of a choreography instance may be the result of a message received from a service provider. The consequent action, as described in the choreography instance, could be to forward the unchanged message to the service requester. The DIP ontology for describing Web service choreographies is available to the DIP consortium only in the deliverable D3.8 [11].

One required interface is exposed for registering choreographies and updating choreography instances with messages - **ChoreographyEngine**.

### 4.10 Parser

The **Parser** parses incoming WSML messages to convert them into the internal data representation for the DIP architecture. The architecture prototype uses the WSMO4j Java class model for its internal data representation so in this case, the parser validates the WSML message and, if everything is ok, creates the corresponding WSMO4j objects.

The required interface for this component is - **Parser**.

### 4.11 Orchestration

The functionality and interface for the DIP Orchestration are not defined in this version of the architecture. However the DIP ontology for describing Web service orchestration is available to the DIP consortium only in the deliverable D3.8 [11]. The final version of the DIP Architecture and Information Model deliverables will contain the details of the orchestration component.

### 4.12 WSMLReasoner

The **WSML Reasoner** is required by several other components in the architecture, notably discovery and both the process and data mediator components. The benefit of annotating Web service and Goal descriptions semantically is realized through the use of a logical reasoner. The Reasoner must allow the knowledge base it operates on to be made available and must accept queries against this knowledge based. The current draft release allows for hierarchical queries on concepts, such as requests for subconcepts or superconcepts, entailment and, some support for queries against the knowledge base available to the Reasoner.

The required interface for this component is - **WSMLReasoner**.
4.13 External Systems

The DIP architecture links with two external systems, WSMO Studio and the Adapter Framework. WSMO Studio provides a GUI to support the creation of WSMO descriptions and access to the repository provided by the DIP architecture. The Adapter Framework is used to link the architecture with external systems that do not exchange messages expressed in WSML at their interfaces.

4.13.1 WSMO Studio

The WSMO Studio prototype is developed as part of work package 4. It offers interface tools for adding, modifying and deleting WSMO descriptions and provides a front-end to connect to and manage WSMO repositories.

4.13.2 Adapter Framework

The Adapter Framework on the left of Figure 4.1 is not part of the internal architecture for DIP but is essential for supporting external applications, that do not directly support WSML, but that may wish to use DIP for goal-based Semantic Web service execution. For example, an adapter may be required to bridge between the data at the interface of an SAP ERP back-end system to transform it to and from WSML that could be passed to DIP for processing. Enterprise Resource Planning systems (ERP) aim to provide a single coherent software system for all functions of an enterprise. For example, an ERP system usually contains module for finance, warehousing, human resources, customer relationship management and so on. SAP is one of the most well-known providers of ERP systems and it is unlikely that they would support a specification such as WSML unless it gains wide industry support.

4.14 DIP API

The DIP API specifies the methods in the interfaces that each of the functional components comprising the DIP architecture must implement. The API is defined using Java and is provided as open-source as part of the WSMX project. TheEntryPoint interface defined in this API is implemented by the WSMX prototype and the IRS-III Semantic Web services execution environment. This provides the basis for system-level interoperability between for two goal-driven Semantic Web service systems.

The DIP API documentation can be found at:

https://bscw.dip.deri.ie/bscw/bscw.cgi/0/63623 and
5 Behavioural View

This section describes a behavioural viewpoint on the DIP architecture from the perspective of a goal-based service invocation. Other behaviours are possible for the DIP architecture without the necessity for redesigning the Core component or the functional components. This is made possible by using definitions of execution semantics to separate the description of the operation of the DIP architecture from the design and implementation of the individual components. Execution semantics are the formal definitions of particular operation scenarios for the architecture. Four mandatory execution semantics are defined for the DIP architecture. The mechanism for describing execution semantics and the four mandatory execution semantics defined for the DIP architecture are the subject of deliverable D6.12. In this section we provide a high level overview of the most important scenario - goal based service invocation. We provide UML2 activity diagrams to describe two variations on the scenario - one-way goal execution and conversation-based goal execution.

1. One way goal execution. All inputs (WSML) required for the service invocation are sent to the DIP system with the Goal definition (WSML).

2. Conversation-based goal execution. Not all inputs are provided up front and there is an over-and-back conversation between the client and the service with the DIP architecture acting in the role of a mediator.

This viewpoint is targeted at the following stakeholders: acquirer of the DIP system, developers, component owners. It may also be of interest to the case-study partners. The viewpoint addresses the following concerns:

- Does the architecture achieve what is required of it in the DIP Project Description of Work?
- Is there sufficient information and detail in the description to progress to a detailed technical design?
- How are individual technical components (e.g. prototypes from the technical work packages 1-5) plugged into the architecture?

In the DIP architecture, communication between components takes place through event-based messaging. Events are raised by messages sent from individual components. The Core component includes a mechanism to accept events, determine what action needs to be taken and carry out this action. For example, it often is the case that an event results in a message being dispatched to another component. The DIP architecture itself adopts the service-oriented architecture style. Each component in the architecture is modelled as a service (sometimes referred to as middleware services to distinguish them from the Web services outside the architecture). This design enables the practice of implementation-hiding, in which the implementation of a service does not need to be known by the client of the service. The aim is to increase flexibility, extensibility and reusability. In particular it facilitates the addition, upgrade or removal of functional components while at the same time maintaining the stability of the system. It facilitates the addition of new components to the DIP architecture that
were not considered during the initial design and the upgrading or removal of others without requiring a re-build of the entire system.

The rest of this section is organised as follow. An informal overview of the relationship between execution semantics and the messaging mechanism in DIP is provided in section 5.1. Section 5.2 provides a UML2 activity diagram and brief description of the one-way goal execution scenario. Section 5.3 describes in more detail the conversation-based Goal execution scenario. Section 5.4 provides a brief description of the participation of the WSMX prototype for the DIP architecture in the ongoing Semantic Web Services Challenge\(^1\).

5.1 Informal Overview of Execution Semantics and Messaging in DIP

Execution semantics are intended to specify system behavior formally and unambiguously and enables its simulation and analysis prior to the execution. The DIP architecture is based on an event-driven architecture composed of loosely coupled components which facilitates the creation of various execution semantics for a system since the activities of the components are stimulated by events as they occur and there are no fixed bindings between components. Components can create or consume messages but they cannot invoke each other directly (strictly speaking, only the core component can interact with other components).

Each execution semantics can be considered to represent a process. Whatever tool is used for the definition of the process must make it possible to verify certain properties of the model; it must check some simple properties such as syntactical correctness, unreachable states or unsatisfiable conditions. Services are requested by using services interfaces. Figure 5.1 depicts a simplified process for goal-based discovery and subsequent invocation of a Web service. The ovals in the diagram represent the steps in the execution semantics for this definition of the discovery process.

Each component has associated configuration metadata which allows the Core to pick up the component and generate wrappers for each component as it is deployed. Figure 5.2 shows the introduction of wrappers and how it is these wrappers that are invoked by the implementation of the execution semantics rather than the components themselves. The wrappers are generated and managed by the Core to separate components from the transport layer for events. Wrappers isolate the component implementation from the execution engine in the Core. One wrapper raises an event with some message content and another wrapper can at some point in time consume this event and react to it. Developers of individual components do not need to be concerned about how this happens. This is significant in that the design and development of functional components need only a minimal knowledge of the messaging mechanism for DIP. From their perspective, they appear to directly invoke other components even though the architecture takes over all responsibility for messaging. Components never need to be explicitly aware of the wrappers and never invoke them directly.

Finally, figure 5.3 depicts how components are decoupled from the process (described in the execution semantics). The deployment of any new execution semantics will remain transparent to components. Based on an execution semantics definition,

\(^1\)http://sws-challenge.org/
Figure 5.1: Sample Abstract Execution Semantics Definition

Figure 5.2: Process, its wrappers and context
these wrappers will only be able to consume and produce particular types of events. In a running system dynamic execution semantics are achieved by mapping abstract system behavior into real event infrastructure of the system.

5.2 One-way Goal Execution Scenario

The service requester, which expects DIP to discover and invoke a Semantic Web service without exchanging additional messages can use this entry-point by providing the goal description, and optionally, the input data required to achieve the goal, both expressed in terms of WSML.

Note: Both UML diagrams below have been created with the community edition of the UML tool from Visual Paradigm. Consequently, there are watermarks on both figures 5.4 and 5.5 which lessens their clarity. A license for the professional edition of this software is being obtained which will enable the removal of these marks.

In figure 5.4, there are two swimlanes to indicate the activities happening inside and outside the DIP architecture. The scenario starts when a Goal and Input data are made available to the system. The first step is to parse the external WSML representation of the Goal and data into the internal WSMO4j representation. Discovery is responsible for finding one or more Web services whose capability matches the Goal. Selection picks one service from the list of those discovered based on the algorithm used by the Selector functional component. Where required data and process mediation take place between the description of the Goal and the description of the Web service. It is possible that neither is required - this is indicated by the note "Can be null-op". The final activity is that a message is sent by the system to the discovered service.
5.3 Conversation-based Goal Execution Scenario

The previous scenario assumed that all messages required by the discovered Web service could be supplied up-front by the service requester sending the Goal. In this scenario shown in figure 5.5, we drop that assumption. There may be multiple messages exchanged between the client application sending the Goal and the Web service. The activity starts when a Goal and the message with the initial service requester input is made available. The activity stops when neither the Web service nor the service requester have any more messages to send.

5.4 SWS Challenge

As a side-note, the WSMX prototype implementation for the DIP architecture is participating in the ongoing SWS Challenge sponsored by DIP. The challenge is organized in a series of phases, the first of these deals with the problems of data and process mediation in the context of a fictitious trading company called Moon. Moon uses two back-end systems to manage its order processing business. The first is a customer relationship management system and the second is an order management system. The challenge provides both of these systems as publicly accessible Web services described using WSDL. The scenario describes how Moon has signed agreements to exchange purchase order messages with its client using the RosettaNet PIP3A4 eBusiness standard. The task for the challenge entrants is to use Semantic Web service technology to mediate between the RosettaNet standard and the data schema used by Moon and to ensure the message exchange between all parties is correctly choreographed. Data mediation is involved in mapping the datatypes of the RosettaNet messages to the datatypes of the back-end systems. Process mediation is involved in mapping the message exchanges defined by the RosettaNet PIP3A4 process to those defined in the WSDL of the back-end systems.
More detail on the WSMX prototype submission to the SWS Challenge can be found in the paper submitted to the Challenge workshop [8]. This paper includes a UML sequence diagram that shows the detailed message flow through the various DIP functional components in the context of the Challenge problem and is relevant to the conversation-based Goal execution activity shown in figure 5.5.
6 SUMMARY

In this document we described the architecture of the current version of the DIP architecture. The document follows the recommendations on using architectural views of IEEE Recommended Practice for Architectural Descriptions of Software-intensive systems. The deliverable identifies stakeholders in the architecture and concerns that these stakeholders are likely to have. Four viewpoints, or perspectives on the architecture are presented. In each case, the stakeholders and concerns the description of the viewpoint addresses, is made clear.

We focused on how the DIP architecture has evolved as a Service Oriented Architecture (SOA) from earlier versions. The adoption of the principles of component decoupling and standardized component interface definitions provide DIP with flexibility as the implementations of functional components are decoupled from the definition of the possible behaviour (execution semantics) of the system.

In particular, in the structural viewpoint of the DIP architecture components, we have provided an overview diagram of the architecture as well as a UML component diagram showing each components’ interfaces and dependencies. In the section describing the behavioural viewpoint we discussed the benefits of separating the functional and behavioural description of a system. UML activity diagrams were used to describe two variations on goal-based invocation using the DIP architecture.

The API for the DIP architecture is documented based on the work in the open-source WSMX project which is a prototype implementation for DIP. The API is documented using Javadoc based on the Java definitions of the component and system interfaces. A zip file containing the API has been submitted to the BSCW document management system used by DIP and to the WSMX prototype homepage and is available through the link included in the relevant section of this deliverable.

In previous versions of this deliverable, there was a more detailed description of the execution semantics for the DIP architecture. In line with agreement from the DIP project management, this description is now aligned with the work in the OASIS Semantic Execution Environment Technical Committee and is the subject of the DIP deliverable, D6.12.
REFERENCES


