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

Deliverable

D6.1
Report on requirements analysis and state-of-the-art

Michael Altenhofen
Manfred Hauswirth
Vesselin Kirov
Atanas Kiryakov
Christoph Mack
Joachim Quantz
Roman Schmidt

28 June 2004
**EXECUTIVE SUMMARY**

This deliverable analyses the state-of-the-art of web service architecture standards (both from industry and from standard bodies) and describes the requirements of DIP with respect to overall architecture and interoperability. The requirements take into account the needs of DIP beyond existing architectures and proposals and the deliverable specifically addresses the requirements defined by the three case studies of DIP (Virtual ISP, eGovernment, and eBanking). These requirements have been generalized to be also helpful to other projects in the semantic web services (SWS) domain. The architectural requirements presented can be viewed as an umbrella integrating the state-of-the-art and requirements analyses done by the other technical workpackages (WP1–WP5) with respect to architecture.

This deliverable contributes to the major DIP result of defining an open source SWS architecture and thus has a major impact on achieving the goals of the DIP project.
Report on requirements analysis and state-of-the-art

Document Information

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

Deliverable Number 6.1 Title Report on requirements analysis and state-of-the-art
Work package Number 6 Title Interoperability and Architecture

Date of delivery Contractual M 6 Actual 28-Jun-04
Status Version. 1.0 final
Nature Prototype Report Dissemination
Dissemination Level Public Consortium

Authors (Partner) Michael Altenhofen (SAP), Manfred Hauswirth (EPFL), Vesselin Kirov (Sirma), Atanas Kiryakov (Sirma), Christoph Mack (Tiscali), Joachim Quantz (inubit), Roman Schmidt (EPFL)

Responsible Author Manfred Hauswirth Email Manfred.Hauswirth@epfl.ch
Partner EPFL Phone +41 21 639 5237

Abstract (for dissemination) This deliverable analyses the state-of-the-art of web service architecture standards (both from industry and from standard bodies) and describes the requirements of DIP with respect to overall architecture and interoperability taking into account specifically (generalized) DIP case studies requirements and requirements beyond the state-of-the-art. The deliverable can be viewed as an umbrella, integrating the state-of-the-art and requirements analyses done by the other technical workpackages (WP1–WP5) with respect to architecture and contributes to the major DIP result of defining an open source SWS architecture and thus has a major impact on achieving the goals of the DIP project.

Keywords SWS architectures, SWS interoperability, SWS architectural requirements
Report on requirements analysis and state-of-the-art

<table>
<thead>
<tr>
<th>Version Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Date</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Deliverable 6.1
# Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway | NUIG | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
Email: chris.bussler@deri.ie  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. Bankinter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmminez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32  
10117 Berlin,  
Germany  
Email: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc. | BT | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham,  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlett,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
Email: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe,  
Germany  
Email: abecker@fzi.de |
<table>
<thead>
<tr>
<th>Organization</th>
<th>Contact Person</th>
<th>Address</th>
<th>Email</th>
<th>Tel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institut für Informatik, Leopold-Franzens Universität Innsbruck</td>
<td>Prof. Dieter Fensel</td>
<td>Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: <a href="mailto:dieter.fensel@uibk.ac.at">dieter.fensel@uibk.ac.at</a> Tel: +43 512 5076485</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILOG SA</td>
<td>Christian de Sainte Marie</td>
<td>9 Rue de Verduc 94253 Gentilly, France Email: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a> Tel: +33 1 49082981</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inubit AG</td>
<td>Torsten Schmale,</td>
<td>Lützowstraße 105-106 D-10785 Berlin, Germany Email: <a href="mailto:ts@inubit.com">ts@inubit.com</a> Tel: +49 30726112 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>Dr. V. Richard Benjamins,</td>
<td>Pedro de Valdivia 10 28006 Madrid, Spain Email: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a> Tel: +34 913 349 797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net Dynamics Internet Technologies GmbH u. Co KG</td>
<td>Peter Smolle</td>
<td>Prinz-Eugen-Strasse 68-70 A-1040 Wien, Austria Email: <a href="mailto:peter.smale@netdynamics-tech.com">peter.smale@netdynamics-tech.com</a> Tel.: +43 1 503982615</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Open University</td>
<td>Dr. John Domingue</td>
<td>Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom Email: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a> Tel.: +44 1908 655014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAP AG</td>
<td>Dr. Elmar Dorner</td>
<td>SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a> Tel: +49 721 6902 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov,</td>
<td>135 Tzarigradsko Chausse, Sofia 1784, Bulgaria Email: <a href="mailto:atanaskiryakov@sirma.bg">atanaskiryakov@sirma.bg</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Report on requirements analysis and state-of-the-art**

Deliverable 6.1
<table>
<thead>
<tr>
<th>Tiscali Österreich GmbH</th>
<th>Tiscali</th>
<th>Tel.: +359 2 9768 303</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tiscali Österreich GmbH</td>
<td>Dieter Haacker</td>
</tr>
<tr>
<td></td>
<td>Diefenbachgasse 35, A-1150 Vienna, Austria</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Dieter.Haacker@at.tiscali.com">Dieter.Haacker@at.tiscali.com</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: +43 1 899 33 160</td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn</td>
<td>Jeff Eisenberg</td>
</tr>
<tr>
<td></td>
<td>Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel.: +972 2 6491111</td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>VUB</td>
<td>Carlo Wouters, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:carlo.wouters@vub.ac.be">carlo.wouters@vub.ac.be</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel.: +32 (0) 2 629 3719</td>
<td></td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

EXECUTIVE SUMMARY ........................................................................................................... I

TABLE OF CONTENTS ........................................................................................................ VII

1 INTRODUCTION .................................................................................................................. 1

2 STATE-OF-THE-ART ......................................................................................................... 2

   2.1 Web Services Architecture (W3C) ............................................................................... 3

      2.1.1 Web Services Architecture Usage Scenarios ....................................................... 4

      2.1.2 Web Services Architecture Requirements ........................................................ 4

   2.2 Global XML Architecture ............................................................................................ 4

   2.3 Web Service Description Language ............................................................................ 5

   2.4 Web Service Modeling Framework .............................................................................. 7

   2.5 Choreography and Workflow ...................................................................................... 7

      2.5.1 Web Service Choreography Description Language ............................................... 8

      2.5.2 WS-BPEL ............................................................................................................... 8

   2.6 UDDI ............................................................................................................................ 10

      2.6.1 Semantic extensions ............................................................................................ 11

      2.6.2 UDDI/OWL-S Matchmaker ............................................................................... 12

   2.7 ebXML ........................................................................................................................ 12

   2.8 WSMO/WSML ............................................................................................................ 12

   2.9 OWL-S ........................................................................................................................ 13

3 CASE STUDY REQUIREMENTS ....................................................................................... 14

   3.1 Specific requirements of the VISP case study ............................................................... 14

      3.1.1 Case study description ......................................................................................... 14

      3.1.2 Use cases ............................................................................................................. 15

      3.1.3 Requirements ...................................................................................................... 17

   3.2 Specific requirements of the eGovernment case study ................................................. 18

      3.2.1 Case study description ......................................................................................... 18

      3.2.2 Use cases ............................................................................................................. 19

      3.2.3 Requirements ...................................................................................................... 20

   3.3 Specific requirements of the e-banking case study ...................................................... 22

      3.3.1 Case study description ......................................................................................... 22

      3.3.2 Use cases ............................................................................................................. 22

      3.3.3 Requirements ...................................................................................................... 23

4 FUNCTIONAL REQUIREMENTS ......................................................................................... 25

   4.1 Composition ................................................................................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Requirements</td>
<td>25</td>
</tr>
<tr>
<td>4.1.2 State of the Art</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Discovery</td>
<td>26</td>
</tr>
<tr>
<td>4.2.1 Requirements</td>
<td>26</td>
</tr>
<tr>
<td>4.2.2 State of the Art</td>
<td>27</td>
</tr>
<tr>
<td>4.3 Invocation</td>
<td>28</td>
</tr>
<tr>
<td>4.3.1 Requirements</td>
<td>28</td>
</tr>
<tr>
<td>4.3.2 State of the Art</td>
<td>28</td>
</tr>
<tr>
<td>4.4 Mediation</td>
<td>30</td>
</tr>
<tr>
<td>4.4.1 Requirements</td>
<td>30</td>
</tr>
<tr>
<td>4.4.2 State of the Art</td>
<td>31</td>
</tr>
<tr>
<td>4.5 Ontology</td>
<td>32</td>
</tr>
<tr>
<td>4.5.1 Requirements</td>
<td>32</td>
</tr>
<tr>
<td>4.5.2 State of the Art</td>
<td>34</td>
</tr>
<tr>
<td>4.6 Publishing</td>
<td>35</td>
</tr>
<tr>
<td>4.6.1 Requirements</td>
<td>35</td>
</tr>
<tr>
<td>4.6.2 State of the Art</td>
<td>36</td>
</tr>
<tr>
<td>4.7 Registry</td>
<td>37</td>
</tr>
<tr>
<td>4.7.1 Requirements</td>
<td>37</td>
</tr>
<tr>
<td>4.7.2 State of the Art</td>
<td>37</td>
</tr>
<tr>
<td>4.8 SWS Description</td>
<td>38</td>
</tr>
<tr>
<td>4.8.1 Requirements</td>
<td>38</td>
</tr>
<tr>
<td>4.8.2 State of the Art</td>
<td>38</td>
</tr>
<tr>
<td>5 NON-FUNCTIONAL REQUIREMENTS</td>
<td>39</td>
</tr>
<tr>
<td>5.1 Trust</td>
<td>39</td>
</tr>
<tr>
<td>5.1.1 Requirements</td>
<td>39</td>
</tr>
<tr>
<td>5.1.2 State of the Art</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Security</td>
<td>41</td>
</tr>
<tr>
<td>5.2.1 Requirements</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2 State of the Art</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2.1 XML Security Base</td>
<td>42</td>
</tr>
<tr>
<td>5.2.2.2 The IBM/Microsoft WS-* series of proposals</td>
<td>43</td>
</tr>
<tr>
<td>5.2.2.3 SAML (Security Assertion Markup Language)</td>
<td>44</td>
</tr>
<tr>
<td>5.2.2.4 XACML (eXtensible Access Control Markup Language)</td>
<td>44</td>
</tr>
<tr>
<td>5.3 Transition path to semantic web services</td>
<td>45</td>
</tr>
<tr>
<td>5.3.1 Non-Functional Requirements related to the usage of ontologies</td>
<td>45</td>
</tr>
<tr>
<td>5.3.2 Changes of the non-functional requirements of the web services</td>
<td>46</td>
</tr>
<tr>
<td>6 INTEROPERABILITY</td>
<td>47</td>
</tr>
<tr>
<td>6.1 Interoperability Problems in Web Services</td>
<td>47</td>
</tr>
</tbody>
</table>
6.2 WS-I (Web Services Interoperability Organization) .................................................. 48
  6.2.1 WS-I Basic Profile 1.0 ................................................................................... 48
  6.2.2 Applying WS-I Basic Profile ......................................................................... 49
  6.3 Interoperability Recommendation for DIP .......................................................... 50

7 CONCLUSIONS ........................................................................................................... 51

REFERENCES .................................................................................................................. 52
1 INTRODUCTION

This deliverable analyses the state-of-the-art of web service architecture standards (both from industry and from standard bodies) and describes the requirements of DIP in respect to overall architecture and interoperability. It provides an overview of the current state-of-the-art of web service architectures and proposals, their main components, abstractions, concepts, and interactions models. As DIP extends the concept of (plain) web services with semantics it has to take into account not only these existing architectures and requirements and extend them with new functionality, components, abstractions, and interaction models that specifically address the requirements of semantic web services and the users of such enhanced services.

To meet these goals all technical workpackages of DIP (WP1-WP6) have provided state-of-the-art and requirements analyses addressing their specific task. WP6 is concerned with DIP’s overall architecture and interoperability and thus can be seen as an umbrella, integrating the state-of-the-art and requirements analyses done by the other technical workpackages (WP1–WP5) with respect to the focus of WP6. In this document we will explicitly refer to the other workpackages for detailed discussions, where relevant, to avoid unnecessary duplication and inconsistencies. The other technical workpackages (WP1-WP5) on the other hand have provided input to this deliverable to make it meet their architectural and interoperability requirements. These requirements have been combined with the requirements of the case studies to steer the process of defining DIP’s overall requirements.

This document is structured as follows: Section 2 overviews the state-of-the-art of web service architectures and web service standards; Section 3 presents the DIP case studies and their relevant requirements on architecture and interoperability for DIP. Section 4 presents the functional requirements and state-of-the-art for the main components followed by the non-functional requirements in Section 5; and finally Section 6 discusses interoperability issues.
2 STATE-OF-THE -ART

This chapter gives a detailed overview of the state-of-the-art of current non-semantic architectures for web services and discusses recent attempts for introducing semantics into such architectures. The current standards and architectures to a large extent follow the model shown in Figure 1.

In the following we will first discuss the two most influential architectural proposals, i.e., Web Service Architecture (WSA) and Global XML Architecture (GXA), which provide a general framework for web service developers. Both architectures are basically based on the same standards but published by different organizations. Other architectures are either of minor relevance or can be seen as variants of these architectures. Then we will present currently available standards for the components in Figure 1. To ensure interoperability and to provide a transition path from non-semantic to semantic web services the requirements defined in these standards have to be evaluated for applicability in the course of the DIP project.

We do not discuss all standards in this section, though, because some standards in our opinion have to be followed implicitly. For example, DIP will certainly use the SOAP messaging standards to be interoperable with existing web service infrastructures. Rather we will focus on interoperability issues which come into play even if the SOAP standard is used (see the discussion in Section 6). Standards, such as security related standards will be discussed in the respective sections of this document dealing with certain architectural aspects such as security explicitly. And finally, other standards will be discussed in the state-of-the-art sections of the individual components in Section 4.

The strategy we decided to follow in the presentation of the state-of-the-art was to present the core web service standards in this section in order to provide a comprehensive overview of the domain and then address the more specific standards in the dedicated sections. We hope that this simplifies the understanding of this document.
by providing the “big picture” first and putting special issues in direct and close context to the components and strategies we plan to use in DIP.

2.1 Web Services Architecture (W3C)

The Web Services Architecture document (WSA) [52] by the W3C intends to provide a common definition of a web service, and defines its place within a larger web services framework. The WSA provides a conceptual model and a context for understanding web services and the relationships between the components of its architectural model. The architecture does not attempt to specify how web services are implemented, and imposes no restriction on how web services might be combined. Instead the WSA describes both the minimal characteristics that are common to all web services, and a number of characteristics that are needed by many, but not all, web services.

Figure 2 shows WSA’s architecture. It consists of requesters interacting with providers via agents implementing a web service defined by Web Service Descriptions (WSD) and semantics. A requester is a person or organization that wants to use a service of a provider offering some functionality. Agents implement web services and interact between the two parties. Several agents can implement the same service but an agent always belongs to one service. The WSD is a machine-processable specification of the web service's interface, written in WSDL. It defines the message formats, data types, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. The semantics of a web service is the shared expectation about the behavior of the service, in particular in response to messages that are sent to it. In effect, this is the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated.

![Diagram of Web Services Architecture](image_url)

Figure 2: Web Services Architecture (W3C)
Figure 2 also shows the interaction model between the two parties: (1) the requester and the provider first become known to each other (e.g., by some directory service); (2) the requester and provider entities agree on the service description and semantics that will govern the interaction between the requester and provider agents; (3) the service description and semantics are realized by the requester and provider agents; and (4) the requester and provider agents exchange messages, thus performing some task on behalf of the requester and provider entities.

The WSA does not define how requesters and providers become known to each other or how they agree on the service descriptions and semantics. It just defines the components and interactions required for web service architectures. Regarding semantics, the central goal of DIP, WSA provides a starting point but no input with respect to DIP’s goal of providing enabling mechanisms, models, and infrastructures for semantic web services.

2.1.1 Web Services Architecture Usage Scenarios

[54] provides a collection of use cases and usage scenarios which illustrate the use of web services. The use cases are used to come up with requirements for web services architectures, as well as to evaluate existing technologies.

2.1.2 Web Services Architecture Requirements

[53] describes a set of requirements for a standard reference architecture for web services developed by the Web Services Architecture Working Group. These requirements are intended to guide the development of the reference architecture and provide a set of measurable constraints on web service implementations by which conformance can be determined. The working group has determined that at the highest level, architectural goals fall into 7 categories:

- **Interoperability**: The WSA should enable the development of interoperable web services across a wide array of environments.
- **Reliability**: The WSA must be reliable and stable over time.
- **Integration** with the World Wide Web: The WSA must be consistent with the current and future evolution of the World Wide Web.
- **Security**: The WSA must provide a secure environment for online processes, as distinct to developing trust between entities.
- **Scalability and Extensibility**: The WSA must enable implementations that are scalable and extensible.
- **Team Goals**: The Web Services Architecture Working Group will work to ensure that the Architecture will meet the needs of the user community.
- **Management and Provisioning**: The standard reference architecture for web services must provide for a manageable, accountable environment for web services operations.

2.2 Global XML Architecture

The Global XML Architecture (GXA) [36] shown in Figure 3 is a layered architecture which builds on the standard web service specifications of Simple Object Access Protocol (SOAP), Universal Data Discovery Interface (UDDI), and Web Services
Description Language (WSDL). The term GXA seems to be used uniquely in the Microsoft context, whereas others refer to these standards as the WS-x family of web services standards.

On top of this base layer more advanced functionalities which are required by any distributed information system are provided, for example, WS-Security or WS-Attachments. The rational is to provide standard protocols at this layer that web services would otherwise have to implement themselves and possibly introduce incompatibilities. The protocols are primarily concerned with the content and structure of individual messages. For example, the Security specification describes (among other things) how security information, such as a Kerberos ticket, is to be embedded into a SOAP message.

The top layer is composed of infrastructure protocols that enable functionalities such as transactions or reliable messaging. Infrastructure protocols describe how sequences of messages are put together to solve a certain problems (to some extent they model simple workflows). The transaction specification, for example, describes the flow of SOAP messages among a set of web services that need to work together to coordinate a series of database updates.

Some of the GXA protocols are already defined or partially implemented (for example, WS-Security and WS-Transaction).

![GXA architecture](image)

2.3 Web Service Description Language

The Web Service Definition Language (WSDL) is a standardization effort of the World Wide Web Consortium to define a model and an XML format which can be used to describe web services in an abstract and (programming) language-neutral way. By specifying the operations that a web service provides and the set of messages that are exchanged during the course of operation, WSDL describes the interface of a service (“what”) without imposing a particular implementation (“how”). In that sense, WSDL shares some similarities with interface definition languages in traditional middleware platforms, like, e.g., CORBA [38], or COM+ [35].

In WSDL, a web service description is split up into two parts [1]: The abstract part specifies the service interface, whereas the concrete part deals with binding these
abstract interfaces to particular message encodings and transport protocols and identifying concrete service access points.

At the highest level, the abstract part consists of three elements: types, messages, and port types. The type element contains the data type definitions in XML Schema that are used for the exchanged messages. These messages, in turn, are listed in the messages part, where each message is defined as a typed document consisting of different parts. Port types are then used to bundle operations and define the set of operations the web service supports. An operation is specified by a set of messages and may belong to one of the following interaction types: A one-way operation contains only one message, which is sent from the client to the service. In a notification operation, a message is sent by the service to the client. Request-response and solicit-response operations consist of two messages. In the first interaction type, the communication is initiated by the client, whereas the second one is started by the service.

The concrete part is composed of bindings and services. A binding takes the operations and messages, which are defined in an abstract port type, and maps them to a particular transport protocol and on-the-wire format. A service is a logical group of ports. A port, in turn, links a binding to a URI (a network address) that can be used to access the implementation of a port type.

The following example is an excerpt of a typical WSDL specification taken from [3]:

```xml
<definitions name="Procurement">
  <message name="OrderMsg">
    <part name="productName" type="xsd:string"/>
    <part name="quantity" type="xsd:integer"/>
  </message>
  <portType name="ProcurementPortType">
    <operation name="OrderGood">
      <input message="OrderMsg"/>
    </operation>
  </portType>

  <binding name="ProcurementSoapBinding" type="ProcurementPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="OrderGood">
      <input><soap:body use="literal"/></input>
    </operation>
  </binding>
  <service name="ProcurementService">
    <port name="ProcurementPort" binding="ProcurementSoapBinding">
      <soap:address location="http://example.com/procurement"/>
    </port>
  </service>
</definitions>
```

It specifies a fictional procurement service that accepts only one one-way message: A client can order a good by specifying the good’s name and the quantity the client wants to order. Since all parts of the message use built-in data types, no additional data type definitions are necessary. This web service is accessible via HTTP and all client requests are sent to the service using the SOAP protocol where the messages are encoded by literally copying the message parts into the SOAP message body.

So far, we have described the WSDL specification according to the version 1.1 draft of the standard. There is currently work going on to define a new version [59]. This new version has a number of interesting features with regard to semantic web services. First
of all, it introduces the concept of *message exchange patterns*. While the version 1.1 specification identifies the meaning of an operation implicitly via the order of the messages that it uses, the new specification introduces new notational elements to explicitly state how a message group should be interpreted. [60] defines a set of predefined message exchange patterns. Furthermore, WSDL2.0 does not limit type definitions to XML Schema, but provides a way to import and use other XML-based schema languages. The Semantic Web Services Interest Group is currently discussing ways in which to support semantic web type systems, like OWL and/or RDFS [45] in WSDL and how such semantic information could be used, e.g., to query, discover, or infer services.

### 2.4 Web Service Modeling Framework

WSMF is an attempt to provide a full modeling framework for building a semantic web service infrastructure [26]. The two main principles of the framework are strong decoupling of various components that comprise the web service application and a powerful mediation service that support interaction of components. Mediation, in the context of WSMF, means mediation among terminologies and mediation between different interaction styles.

WSMF consists of four elements [26]: *ontologies* that provide the terminology used by the other elements; *goal repositories* that define the problems to be solved by the web services; *web service descriptions*; and *mediators* enabling interoperability.

Originally, WSMF explicitly abstained from defining a concrete syntax or formal semantics of the WSMF. While it states that this is quite straightforward to do, it acknowledges that quite a lot of adaptation work may be required. Since the original publication quite a lot of work has been done regarding the framework. Namely, the work in the SWWS IST project (running since Sept. 2002) has concentrated on implementation of this framework and WSMF is the basis for the Web Service Modeling Ontology initiative (WSMO, see [http://www.wsmo.org](http://www.wsmo.org)). The future development of WSMF is expected to take place within WSMO.

### 2.5 Choreography and Workflow

Web services choreography specifications pick up things from where WSDL’s expressive power ends, i.e., they build on services described in WSDL and define a sequence of interacting services and an information flow.

WSDL itself essentially supports the definition of the static interface of a web service. Operations in a WSDL portType (WSDL container construct for web service interface definition) are grouped into a single interface but they are independent of each other and the WSDL grammar does not permit specification of any relation or required sequencing of operations within a portType. Hence the interaction model of a WSDL portType is stateless and static without any correlation of interactions at the interface defined. Additionally, WSDL describes interfaces from the perspective of a service (provider) and hence it is geared toward a client-server interaction model. Collaborative process models typically involve both client-server and peer-to-peer type interactions.

---

1 The Web Services Modeling Framework should not be confused with HP’s Web Services Management Framework, for which the same acronym, WSMF, is used as well.
with long-running and stateful conversations involving two or more parties, which WSDL is not equipped to deliver. Hence the web services choreography specifications use WSDL as a basis and extend the WSDL functionality to support choreography.

Though there exist several industry specifications that address collaborative process modeling two main specifications, i.e., WS-CDL (Web Service Choreography Description Language) and WS-BPEL (Web Services Business Process Execution Language) are likely to become standards but are incompatible to each other.

### 2.5.1 Web Service Choreography Description Language

The Web Services Choreography Description Language (WS-CDL) [1] is an XML-based language that describes peer-to-peer collaborations of web services participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal. The Web Services Choreography specification is targeted at composing interoperable peer-to-peer collaborations between any type of web service participant regardless of the supporting platform or programming model used by the implementation of the hosting environment.

WS-CDL describes the behavior of a web service in terms of choreographed "activities". Activities may be atomic or complex. Atomic activities represent the basic unit of behavior of a web service, such as sending or receiving a message, or waiting for a specified amount of time. Atomic activities dealing with messages correspond to the execution of operations defined in a portType of a WSDL. Atomic activities are also referred to as "Actions".

Complex activities are recursively composed of other activities; and ultimately, each complex activity is composed of actions. Each complex activity defines a specific kind of choreography for the activities it is composed of. WS-CDL supports the definition of the following types of choreographies:

- **Sequential execution**: The activities must be executed in sequential order.
- **Parallel execution**: All activities must be executed, but they may be executed in any order.
- **Looping**: The activities are repeatedly executed based on the evaluation of a condition or an expression. WSCI supports *for-each*, *while*, and *repeat-until* style loops.
- **Conditional execution**: One out of several sets of activities is executed based on the evaluation of conditions (*switch*) or based on the occurrence of an event (*choice*)

### 2.5.2 WS-BPEL

WS-BPEL [4] defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through web service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a *partner link*. The WS-BPEL process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. WS-BPEL also introduces systematic mechanisms for dealing with business exceptions and processing faults. Finally, WS-BPEL introduces a mechanism
to define how individual or composite activities within a process are to be compensated in cases where exceptions occur or a partner requests a reversal.

WS-BPEL is layered on top of several XML specifications: WSDL 1.1, XML Schema 1.0, and XPath1.0. WSDL messages and XML Schema type definitions provide the data model used by WS-BPEL processes. XPath provides support for data manipulation. All external resources and partners are represented as WSDL services. WS-BPEL provides extensibility to accommodate future versions of these standards, specifically the XPath and related standards used in XML processing.

A business process is defined by four major sections in a process definition:

- The `<variables>` section defines the data variables used by the process, providing their definitions in terms of WSDL message types, XML Schema simple types, or XML Schema elements. Variables allow processes to maintain state data and process history based on messages exchanged.

- The `<partnerLinks>` section defines the different parties that interact with the business process in the course of processing. Each partner link is characterized by a partner link type and a role name. This information identifies the functionality that must be provided by the business process and by the partner service for the relationship to succeed, that is, the portTypes that the process and the partner need to implement.

- The `<faultHandlers>` section contains fault handlers defining the activities that must be performed in response to faults resulting from the invocation of the assessment and approval services. In WS-BPEL, all faults, whether internal or resulting from a service invocation, are identified by a qualified name. In particular, each WSDL fault is identified in WS-BPEL by a qualified name formed by the target namespace of the WSDL document in which the relevant portType and fault are defined, and the name of the fault. It is important to note, however, that because WSDL 1.1 does not require that fault names be unique within the namespace where the operation is defined, all faults sharing a common name and defined in the same namespace are indistinguishable. In spite of this serious WSDL limitation, WS-BPEL provides a uniform naming model for faults, in the expectation that future versions of WSDL will provide a better fault-naming model.

- The rest of the process definition contains the description of the normal behavior for requests. The major elements of this description are: `<receive>`, `<reply>`, `<invoke>`, `<assign>`, `<throw>`, `<terminate>`, `<wait>`, `<empty>`, `<sequence>`, `<switch>`, `<while>`, `<pick>`, `<flow>`, `<scope>`, `<compensate>`.

The following example is taken from [4] and shows a process definition for a shipping example. At first, a value is copied from another variable (message) to be sent in the next invoke statement. After invoking the service defined by partnerLink, portType, operation, and input and output variables a return message is awaited. The surrounding `<sequence>` statement defines that the elements are to be executed sequentially and not simultaneous (<flow>).
2.6 UDDI

UDDI (Universal Discovery Description and Integration) version 3 [50] is a “meta service” for locating web services by enabling robust queries against rich metadata. Expanding on the foundation of versions 1 and 2 of UDDI, version 3 offers the industry a specification for building flexible, interoperable XML web services registries useful in private as well as public deployments. With a vast array of enhancements – including multi-registry topologies, increased security features, improved WSDL support, a new subscription API and core information model advances – the version 3 specification offers clients and implementers a comprehensive and complete blueprint of a description and discovery foundation for a diverse set of web service architectures.

One top-level data structure within UDDI is the businessEntity structure, used to represent businesses and providers within UDDI. It contains descriptive information about the business or provider and about the services it offers. This would include information such as names and descriptions in multiple languages, contact information and classification information. Service descriptions and technical information are expressed within a businessEntity by contained businessService and bindingTemplate structures.

Each businessService structure represents a logical grouping of web services. At the service level, there is still no technical information provided about those services; rather, this structure enables assembling a set of services under a common fabric. Each businessService is the logical child of a single businessEntity. Each businessService contains descriptive information – names, descriptions and classification information – outlining the purpose of the individual web service found within it. For example, a businessService structure could contain a set of Purchase Order web services (submission, confirmation and notification) that are provided by a business.

Each bindingTemplate structure represents an individual web service. In contrast with the businessService and businessEntity structures, which are oriented toward auxiliary
information about providers and services, a bindingTemplate provides the technical information needed by applications to bind and interact with the web service being described. It must contain either the access point for a given service or an indirection mechanism that will lead to the access point. Each binding template is the child of a single businessService. The containing parents, a bindingTemplate can be extended with metadata that enable the discovery of that bindingTemplate, given a set of parameters and criteria.

Technical Models (or tModels) are used in UDDI to represent unique concepts or constructs. They provide a structure that allows re-use and, thus, factorization and standardization within a software framework. The UDDI information model is based on this notion of shared specifications and uses tModels to engender this behavior. For this reason, tModels exist outside the parent-child containment relationships and between the businessEntity, businessService, and bindingTemplate structures. Each distinct specification, transport, protocol, or namespace is represented by a tModel. Examples of tModels that enable the interoperability of web services include those based on Web Service Description Language (WSDL), XML Schema Definition (XSD), and other documents that outline and specify the contract and behavior – i.e., the interface – that a web service may choose to comply with. To describe a web service that conforms to a particular set of specifications, transports, and protocols, references to the tModels that represent these concepts are placed in the bindingTemplate. In such a way, tModels can be re-used by multiple bindingTemplates. The bindingTemplates that refer to precisely the same set of tModels are said to have the same “technical fingerprint” and are of the same type. In this way, tModels can be used to promote the interoperability between software systems.

2.6.1 Semantic extensions

Current efforts at the OASIS UDDI TC concentrate on specifying how to import and manage taxonomies inside the UDDI registry. This is a sharp departure from UDDI version 2 where the external taxonomies were required to live outside of the registry and the taxonomy provider was required to expose a service that would check the consistency of a value from the taxonomy. There are no attempts to base the discovery process on taxonomies or to provide a semantic query interface. Currently the only expected extension for UDDI version 3 is an external taxonomy description language based on OWL which satisfies the taxonomy language requirements of UDDI. The proposal can be found at [39] and [2].

Semantic Service Matchmaker (Carnegie Mellon University and Toshiba Corp.) [29] is a public experiment by Toshiba Corp. and NTT-Communications (www.agent-net.com). The work attempts to extend the existing UDDI infrastructure with a matchmaker agent between the publishing application and the UDDI registry or between the discovery/query agent and the registry. The matchmaker client tools can be used during publishing to semantically annotate a WSDL description and send it to the matchmaker's agent which extracts the semantic annotations, stores them locally and registers the UDDI compliant part in the UDDI registry. The client tools described in the report support annotations in RDFS, DAML and OWL. An agent receives normal UDDI queries that can be augmented with semantic annotations of the desired service interfaces. The agent extracts the annotations and uses them to search for a match in the registered annotations. Then it extracts the detailed information for matching services from the UDDI registry and returns them. Of course, all the registration or query calls
can be performed without any semantic annotations; in which case the matchmaker's agent would work as a simple proxy. Another interesting aspect of the public experiment is that it was performed on the live public UDDI registry maintained by NTT-Communications, one of the four official UDDI operators, to test the scalability of the approach under real-world conditions.

2.6.2 UDDI/OWL-S Matchmaker

[44] augments UDDI registries with an additional semantic layer that enables capability-based matching. Services that advertise using DAML-S (now called OWL-S, see Section 2.9) are also advertised in the UDDI registry, and therefore can be found and retrieved by using UDDI keyword search. The DAML-S/UDDI Translator constructs a UDDI service description using information about the service provider, and the service name. The result of the registration with UDDI is a reference ID of the service. This ID combined with the capability description of the advertisement is sent to the DAML-S Matching Engine that stores the advertisement for capability matching. Requests follow the opposite direction: At first the DAML-S Matchmaker performs the capability matching. The result of the matching is the selected advertisement of the providers and a reference to the UDDI service record. The combination of UDDI records and advertisements is then sent to the requester.

The actual DAML-S based matching engine component selects the advertisements that are relevant for the current request from an advertisement database. Then it uses the DAML+OIL reasoner to compute the level of matching. In turn the DAML+OIL reasoner uses the ontology database data during computing the level of matching. The advertisement database also takes advantage of the ontologies database to index advertisements for fast retrieval at matching time.

2.7 ebXML

Electronic Business XML (ebXML) is an initiative from OASIS and United Nations Centre for Trade Facilitation and Electronic Business. As UDDI does not provide an internal mechanism to store generic service semantics, ebXML aims to provide the exchange of electronic business data in Business-to-Business and Business-to-Customer environments.

An ebXML compliant registry allows metadata to be stored in the registry. This is achieved through a “classification” mechanism, called ClassificationScheme which helps to classify the objects in the registry.

An application of ebXML to web service composition (in fact discovery) is demonstrated in [24] where only simple workflows are taken into account. Further details about ebXML can be found in [22].

2.8 WSMO/WSML

WSMO (http://www.wsmo.org/) is a web service modeling ontology, currently under development. Among the expected outputs of the WSMO project are a meta model and a core model for describing semantic web services, and adequate proof systems (currently Flora2) under the responsibility of the WSML (http://www.wsmo.org/wsml) working group (web service modeling language).
Support for web service composition in WSMO is offered via the notions of goals, capabilities, interfaces, choreographies and orchestrations. The technical choices made by the WSMO group are a key requirement for web service composition in DIP.

2.9 OWL-S

OWL-S (formerly DAML-S: http://www.daml.org/services/owl-s/) is a core set of markup language constructs for describing the properties and capabilities of web services in unambiguous, computer-interpretable form.

OWL-S intends to facilitate the automation of web service tasks including automated web service discovery, execution, interoperaction, composition and execution monitoring. Following the layered approach to markup language development, the current version of OWL-S builds on top of OWL.

A large number of composition related results use DAML-S as a language for representing ontologies, albeit in a rather limited manner (e.g. to describe the ontologies bound to input output data).

Some of the intuitions regarding the current development of WSMO (and not of using OWL-S in DIP) have to do with expected limitations of OWL-S that derive from the choice of using description logic as the logic formalism basis of OWL. The choice of F-logic [30] instead is expected to yield more possibilities.
3 Case Study Requirements

This section briefly presents the three case studies of DIP and their requirements. Detailed descriptions of the case studies are provided by the deliverables of the case study workpackages. Here we focus on the specific requirements that need to be addressed by the DIP architecture to make it a platform the case studies can build on.

3.1 Specific requirements of the VISP case study

3.1.1 Case study description

The Virtual Internet Service Provider (VISP) case study will provide a platform that makes it easy for ISPs to build VISPs according to their individual needs.

To understand the processes and requirements described in the following sections the players and some ISP terminology must first be introduced.

Players

Component suppliers are companies that provide Internet services or any additional services which an ISP needs to provide his offering.

An ISP (Internet Service Provider) is the company that sells all kinds of Internet services. The product portfolio of a typical ISP comprises: IP-connectivity, domains, e-mail services, hosting, housing, security solutions, portal services and hardware. Usually ISPs are the component suppliers for most of their services themselves. An ISP handles all the technical needs of the end user.

A VISP (Virtual Internet Service Provider) is a company that combines Internet services of one or more ISPs into a dedicated product which it sells but without the expenses and duties required in providing those services itself. The typical VISP usually does not invest in the necessary infrastructure, i.e., in any network, equipment or backroom/technical support needed to offer Internet services.

Consumers are placed at the head of the value chain, they consume services from their ISP, which can be either an ISP or a VISP. The important fact here is, that the nature of the ISP is not important and, if a VISP is involved in the chain, sometimes not even visible to the consumer.

ISP terminology

A product describes what is sold to the customer. It comprises one or more services as well as legal details like information about the price, the billing period for recurring charges, the minimum contract period, or the cancellation period. Additionally, the configuration parameters are defined for each service or marked to be supplied by the customer.

An account is an instance of a product that is personalized according to the consumer’s demands.
3.1.2 Use cases

The platform envisioned for the VISP use case shown in Figure 4 mediates communication between the technical systems of the VISP, component suppliers and the internal ISP systems. Figure 4 shows a system with only one component supplier.

![Figure 4: VISP case study](image)

The VISP platform is typically hosted by the ISP. The technical systems of the VISP can either be hosted by the ISP, the VISP itself or some third party company. The platform basically consists of three core components, which are the repository, the registry and the workflow engine. The repository stores the description of component suppliers, VISPs, products, mediators and ontologies. The registry component stores the information about the involved web services. Typically web services will be used to order and maintain products and the consumer’s accounts. The workflow system stores and executes the workflow for the processes described in the next section.

Process Description of a VISP

- **Component Supplier registers Product**: Via a web interface the component supplier product manager can register supplier products and the service interfaces to order and maintain these products in the platform registry. Two kinds of component suppliers have to be distinguished: There are component suppliers who do support and those who do not support SWS.
  - **Component suppliers who do support SWS** have their own product ontology. Once a mediator has been created to map the component supplier’s and the VISP platform’s ontology it can easily be reused for all kinds of products and service interfaces.
When a component supplier who does not support SWS registers his product and service interfaces, the VISP platform has to assist the component supplier in mapping his definitions and parameters to the platform’s ontology. The result of this process is a mediator, which is specifically designed for the component supplier’s product.

**ISP/VISP registers Product:** The VISP product manager can define his products in the platform’s repository. To find the best component supplier products for the realization, the product manager can specify a goal that describes the VISP product he wants to create. The repository is queried with this goal to find component supplier products, which implement the required Internet services. To finally compose the new VISP product the product manager then chooses among the suggested component supplier products. With the creation of a new VISP product the product manager also has to redefine the workflow processes for ordering, maintaining and billing. For these workflows there are template processes, which the product manager has to adopt to implement the specific processes for his new product.

**Consumer orders VISP Product:** The consumer accesses the eCommerce system to order a VISP product. This involves

- checking if the product can be produced in the component supplier provisioning system
- ordering the product in the component supplier CCBS (Customer Care and Billing System)
- ordering the product in the VISP CCBS

**Consumer submits Trouble Ticket:** The consumer reports a problem with their service. The VISP creates a trouble ticket. The VISP assesses the problem and either deals with it directly or escalates it to a component supplier. It finds the appropriate component supplier and interacts with their service assurance system. The process is defined at design time or during maintenance and modification. The designer uses the repository to publish or subscribe to the processes and services.

**Component supplier reports service problem:** The component supplier recognizes that a problem has occurred and creates a trouble ticket in the service assurance system. The component supplier informs the relevant VISP that the problem has occurred. The VISP informs the appropriate consumer. Again, the process is defined at design time or during maintenance and modification. The designer uses the repository to publish or subscribe to processes / services.

**Consumer changes Account Configuration:** Via the Customer Corner the consumer enters the new configuration for his account. The VISP CCBS finds the appropriate component supplier account and checks the registry for the appropriate service to perform the consumer’s request, which is then carried out by the component supplier.

**VISP generates consumer invoice:** On a regular basis the VISP CCBS bills the consumer’s VISP accounts. To obtain the usage data for the associated component supplier accounts the VISP CCBS uses the registry to query the associated usage mediation interface of the component supplier.
3.1.3 Requirements

The following list provides the requirements of the VISP case study:

- **General**: Since it cannot be assumed that all technical systems in the VISP platform environment (especially those of the component suppliers) will have SWS support for the possibility of co-existence of old-style WS and SWS in the DIP architecture is a requirement.

- **Composition**: Composition of services goes hand in hand with process management (provided that the definition of process management can be interpreted bidirectionally). Therefore workflows can be addressed by composition. However, in the VISP case study there is a requirement to build upon existing static business processes. For example, provided that the consumer orders a VISP product (goal) we expect the composition process to initiate all the steps described in the respective ordering process above. For different VISP products different web services on the component supplier side have to be called. Which specific web services are called does not have to be determined during the run time of the ordering process but has already been determined when the VISP product manager created the VISP product.

- **Discovery**: Sophisticated discovery is obviously a requirement for finding the appropriate interfaces (web services) on the component supplier and VISP side and to find the least cost component suppliers for a given VISP product according to predefined rules. The discovery process is restricted to services provided by the component suppliers, which are in a contractual relationships.

- **Invocation**: The VISP Platform will not provide the caller with information how to reach a specified goal but do the work to reach the goal by itself. Therefore a requirement exists for the platform to invoke web services by itself.

- **Mediation**: In the general case the components on the component suppliers’ and the VISP/ISP side do neither share a common process nor data ontology. As Figure 4 shows the VISP platform will have to mediate between these different ontologies.

- **Reasoning**: During the process when the VISP product manager determines which component supplier products are chosen in order to realize a VISP product a requirement for reasoning exists. The idea is that the VISP product manager defines rules to steer the selection of the component suppliers. A very simple rule might be: “The sum of the installation costs of the chosen component supplier products must be a minimum and not higher than 90% of the installation costs of the VISP product”.

- **Compensation**: The ordering of a VISP product which consists of more multiple component suppliers’ products or the processing of an order which comprises more than one VISP product shows the necessity of compensation for the VISP case study. A VISP order (or a VISP product) is usually an atomic entity. If during the process one component cannot be ordered, the ordering of all items that have been ordered in the same context before must be compensated.

- **Monitoring**: Real-time monitoring is possibly not a requirement for the VISP platform at the moment but may be for debugging and legal reasons, for example, to prove that certain actions have been executed. Placing information
in some kind of log file or database is required to be able to reconstruct the reasons why certain actions have or have not been executed and decisions have or have not been made.

- **Registry:** Figure 4 and the processes show a requirement for a registry component inside the VISP architecture. The registry must only be accessible for the ISP and all component suppliers that have an appropriate contract with the ISP (see also the security requirements).

- **Publishing:** The requirement for publishing arises from the first two process examples and is closely linked to the requirement for a registry (see above).

- **Trust:** The VISP platform will operate in a B2B environment with a limited number of business partners. Since the interacting business partners deal with matters of trust in written contracts, trust can be established if the systems of communicating partners in the scenarios described above can be sure to definitely talk to (and only to) the partner system they are addressing.

- **Security:** Security will be a requirement at least on two levels.
  - Level 1 - Message content: Messages will be transferred over the public Internet infrastructure and must not be readable or writable by any unintended communication partner.
  - Level 2 - Platform functionality: According to the role of a communication partner only a limited part of the platform functionality may be offered to the partner, e.g., a component supplier must not browse the platform registry for VISP products or offers of other component suppliers.

### 3.2 Specific requirements of the eGovernment case study

#### 3.2.1 Case study description

Local government authorities in Europe need to develop ways of delivering more usable and comprehensive electronic services to their citizens in a cost-effective way, if they are to be able to comply with the goals of an e-Europe and the related growth in national e-government imperatives. To achieve these goals, they need to understand and have confidence in the application of technologies which can improve electronic service delivery. There is a need to develop and test applications which enable a “supply chain” involving a variety of data and service suppliers, to support joint working and system integration for provision of joined-up services for the citizen and local businesses. Together, these point to a need to move current applications to a web services environment supported by a rich citizen-focused ontology for e-public services to facilitate description, discovery and matching of services.

The initial case study involves development of a citizen’s information service (seamlessUK/Essex Online) in a web services environment. This will require the development and testing of an e-government ontology to support the description and discovery of web services relevant to e-government and public administration. The case study will also implement, test and modify tools produced within DIP in a real world multi-provider e-government environment. In addition, these will also support the integration of enterprise applications within Essex County Council and of back-office systems and processes between the County Council and its partner agencies, necessary
for the delivery of “joined-up” services. In addition, the case study will help understand and develop market conditions to improve take-up and awareness of semantic web services among public administrations in Europe.

More specifically, work in the first phase of DIP will focus on the scenario “Change of Circumstances”, where a citizen changes her address for example. Depending on the citizen’s profile various agencies have to be informed about the change of address and the benefits and services which are eligible after the moving has occurred. This might also include benefits and services for other citizens, e.g., an elderly mother moving in with her daughter might make the daughter eligible for certain benefits.

In a first step, mock-ups will be implemented covering the following basic functionality:

- A web-based front-end allowing citizens or social workers to enter data, e.g. regarding change of address
- The Essex County Council Citizen Server, which
  - Controls the interaction taking place in the front end – portal application
  - Administrates citizen profiles
  - Determines agencies and services to involve
- Simulations of selected agencies and services, which
  - Contain information about citizens which has to be kept up to date
  - Determine eligibility for benefits
- A repository containing citizen profiles

The mock-ups will demonstrate two major characteristics of the eGovernment Case Study:

1. Workflows triggered by a change of circumstances can become rather complex and generally involve several agencies/services in addition to the citizen and the County Council.

2. Agencies/Services use different data models, i.e. communication between the County Council Server and the various agencies requires data mediation.

3.2.2 Use cases

For the initial version the following scenario has been chosen: A part-time employed single woman moving into a new rented house, in the same local authority area as their previous address, in order to look after her disabled 86 year old mother, whose previous address was also in the same local authority area.

This will trigger the following sequence of activities:

1. The mother’s GP informs the case worker for the mother in Community Care of the change of circumstances
2. The mother gives specific permission to her daughter to act with ‘enduring power of attorney’ on her behalf
3. The Case Worker shares change of address details with relevant partner organizations (Housing, Pensions Service) and suppliers of external commissioned services (meals on wheels and nursing support)

4. The Case Worker conducts an assessment and reaches a decision on new entitlements of the woman and her mother to benefits and services

5. The Case worker amends the client records for the woman and her mother on the SWIFT system.

6. With the help of the Case Worker, the woman applies for Housing Benefit and/or Council Tax Benefit

7. Housing Department checks the electoral roll to confirm the address

8. Housing Department notifies the woman and her case worker of eligibility for Housing Benefit and/or Council Tax Benefit

9. The woman asks for a statement of how the benefit was calculated: Housing provides the statement

10. The woman, on behalf of her mother and with the help of the case worker, applies for Attendance Allowance. JobCentre plus communicates the decision to them

11. The woman applies for Carer’s Allowance. Disability and Carers communicate the decision.

12. The woman, on behalf of her mother and with the help of the case worker, applies for Pension Credit from JobCentre plus

13. The woman, with the help of the case worker, applies for ‘Building Regulations’ permission to adapt the house for her mother’s disability. Planning Dept (or someone) informs them of the decision.

14. The woman, with the help of the case worker, applies for an additional discount on Council Tax Benefit. Housing informs them of the decision.

15. Community Care initiates payment for ‘care’ benefits.

16. Housing initiates payment for ‘pure’ housing benefits.

3.2.3 Requirements

There are several requirements concerning system behavior of the production system to be eventually deployed at Essex City Council. These include:

- **Reliability**: Services need to be available seven days a week, 24 hours a day to meet public need.

- **Performance Requirements and Scalability**: Response time should be in the range of average response time offered by web-based services. It is not yet clear how many users will have to be supported concurrently.

- **Security/Trust**: Neither data sharing nor authentication protocols are as yet well established between the agencies concerned in delivering the proposed services.
  - A significant amount of the data to be used is confidential or otherwise sensitive to privacy/data protection laws and ethics. The system would need
to assure this. Essex has a secure virtual private network which links the Boroughs/Districts and the County Council - however at the moment it does not include the external partners, e.g., JobCentre+, Pensions, Police, Health, etc.

- Audit trails are mandatory within Essex County Council.
- Legal transactions, e.g., signing power of attorney, will be necessary within the Case Study

**Platform Support/Portability:** Pilot/prototype services will be run from an independent server outside the County Council firewall in the short-to-medium term. All of the non-County Council organizations involved would currently be wary of letting the Essex County Council through their firewalls. None of the organizations we have interviewed are apparently running any web services applications yet. It has been discussed that the Case Study will initially set up a ‘middleware’ platform to handle semantic web services and then communicate with the individual systems in a more traditional way.

**Installation/Upgrade:** Upgrades should have a minimal impact on the installed system, e.g., it should be possible to keep any persistent data from previous versions.

**Compatibility:** It should be possible to integrate the system into the existing County Council and partner infrastructures, e.g., we want to reuse existing execution environments.

Some of these requirements are clearly outside of the scope of the DIP project as they concern properties to be provided by the production environment. However, these requirements should be taken into account when developing the overall architecture. In particular, it has to be made sure that architecture decisions do not prevent the above requirements to be satisfied.

Additionally, the case study has to meet the requirements of the e-Government Interoperability Framework (eGIF) defined in [9].

With respect to specific DIP technology the following requirements arise in the eGovernment Case Study:

**Data Mediation:** data mediation will be needed, as the various agencies and services to be connected to the County Council Server use different data formats. As the County Council Server will act as a central broker or hub, controlling the interaction between citizens/social workers and agencies/services, a central ontology can be established. Mediation is therefore required between the various formats used at the agencies/services and the central ontology used at the County Council Server. Mediation also has to support non-XML data formats.

**Workflow:** complex workflows have to be supported involving several systems. As DIP has no control over the agencies/services it has to be possible to include non-Web-Service-components in the workflow. It is to be expected that we will not even be allowed to add web service wrappers at the agencies/services, i.e. communication between the County Council Server, which will be supporting
semantic web services, and other agencies/services might have to be based on very traditional communication protocols, e.g., ftp.

3.3 Specific requirements of the e-banking case study

3.3.1 Case study description

In recent years there has been a proliferation of services offered through the Internet. Some of these services are delivered by legacy systems that have been interfaced to the Internet, while other services have been designed with native Internet technology. In the area of e-banking, for example, services have been developed to enable clients to consult their balances, and make financial transactions. Another example of an e-banking service is when banks provide value added services such as real estate information related to mortgages. While e-banking is attracting increasingly more customers (a significant percentage of the European population has an online bank account), we are only starting to see its potential strength for both businesses and consumers. While most financial institutions offer simple straightforward information portals, a few of them are offering advanced services such as, financial aggregators, where distributed, heterogeneous information is aggregated into a one-stop-shop interface. While those applications are quite advanced, their challenge lies in the cost of construction and maintenance.

3.3.2 Use cases

For the initial mock-up the following scenario has been chosen: Juan Esmiz and his girlfriend want to get married and buy a house. They have been searching on the Real Estates market and they have found a lovely flat, near Madrid. The flat costs 300000 Euros and, of course, they do not have that amount of money. So, they need a loan to buy the house:

1. **Information stage**: Juan knows that some Web Portals have their own simulator of loan formulae and comparators of different mortgage offers (from different banks and other financial institutions). He opens with his browser the www.comparator.com webpage. There, he can find out how much he needs to save (every month) to pay 300000 Euros in 20, 25, or 30 years, using the different forms to calculate these values.

2. Once Juan knows that he wants a mortgage loan of 300000 Euros and wants to pay it back within 30 years, he fills in the comparator (another, or the same, form) and the comparator searches for offers that match with these constraints (amount boundary, term in years). The comparator searches online for bank that offer these kinds of mortgages and takes other values needed for calculating the monthly amount (interest rate, time of applicability of each type of interest and commissions).

3. The comparator has gathered different offers from the banks and has calculated what the monthly amount and the interest rate will be. It sorts the offers, using the term called "TAE" (it is determined using the interest rate and the commissions). The comparator gives Juan a list with the most promising (in fact, there is nothing like 'binding offer') mortgage offer. Juan can choose (or not) to ask for a formal mortgage offer with the desired bank. **End of Information Stage** (this stage will be what we will cover in our mock-up)
4. **Request Stage**: Juan contacts with the bank "MockUP Bank" and asks for a mortgage loan. One actor (at the moment a person) on behalf of the bank (the “officer”) takes the main data from Juan (amount of the mortgage, term, incomes of Juan and his girlfriend (if they want to pay together) and other data). A preliminary study of the customer and the property is started. Juan is asked to collect and submit official documents about their incomes and other formal data (about the property).

5. **Data Gathering stage**: The “officer” requests a report about the property from an appraisal entity.

6. The “officer” requests a report about Juan and his girlfriend (if is needed) from a “defaulter registry”.

7. The “officer” inquires the Public Property Registry about the legal situation of the property.

8. Juan collects all the requested data and gives formal testimony about their content and the verity of them.

   **End of Data Gathering Stage.**

9. **Negotiation and Contract Stage**: All the actors involved in the Data Gathering stage deliver their reports to the “officer”.

10. An internal evaluation of the reports suggests (or not) the viability of the mortgage (that is: Juan could pay, or at least the value of the property is enough to grant a benefit).

11. The “officer” sends to Juan a “contract draft”.

12. Juan agrees with the draft.

13. The “officer” asks a Public Notary for an appointment to sign a contract between the bank and Juan in front of him.

14. The Public Notary agrees and makes a reservation on his schedule.

15. The contract is signed.

   **End of the Negotiation and Contract Stage**

16. **Registry Stage**: The “officer” requests from the Public Property Registry an Update of the Property condition.

17. The Public Property Registry updates the Property condition.

   **End of the Registry Stage, of the Request Stage, and of the scope of the case study.**

3.3.3 **Requirements**

There are several requirements including:

- **Composition**: Composition is needed wherever some complex data is gathered from different services which offer portions of the data. Besides, the whole process of the mortgage requesting and examination could be seen as a composition of different subprocesses (appraisal, data gathering, document checking, notarial deeds, register).
- **Discovery**: Discovery is one of the main features needed, mostly when comparing different mortgage offers. These ‘offers’ are discovered at runtime. Several banks offer mortgages under specific constraints and these constraints may be used to discriminate them in a (de-)centralized repository.

- **Invocation**: Several internal calculations and processes will be made inside the mortgage application in order to evaluate, for example, customer acceptance criteria or mortgage calculations. Some of these processes can be performed using external web services, so invocation is a requirement from our platform. Other interesting requirements are the invocation and synchronization of asynchronous services which requires not only execution of SWS but also human interaction.

- **Mediation**: Mediation between different ontologies is definitively a requirement since many public data providers are involved and they do not necessarily use the same ontology. Some process mediation would be also needed if the Service Providers use specific ways to interact with them.

- **Monitoring**: Real-time monitoring is needed in order to follow the current state of the mortgage process for a given customer.

- **Registry**: For contacting the appraisal, notarial and registry components a registry component inside the architecture is definitively necessary. The advertisement of several services by service providers will allow us to discover, select and invoke them.

- **Security**: Two different levels of security can be distinguished:
  - Level 1: Personal data and private data communication. This kind of data (the customer’s data from other financial institutions, notarial information, property information) must be transmitted on a secure channel that provides confidentiality, trust and non-repudiation. Any data of this kind should not be legible if intercepted by a third party.
  - Level 2: Public information concerning banks’ public data, like mortgage characteristics, can be transported using open channels.

- **Trust**: This requirement follows the same guidelines as the previous one. Sensible data must always come from trusted sources. However, different actors could use the application and may be trusted once they provide their login info (it could be necessary that a legal contract between the moneylender and the customer warrants this).
4 FUNCTIONAL REQUIREMENTS

This section defines the functional requirements of DIP. The functional requirements described in [9] are included and not explicitly listed again.

4.1 Composition

Composition refers to the possibility of implementing a web service via subsequent invocation of other web services. Composition can be viewed as a design time issue, in which case, from the Service Usage standpoint it requires using computer-aided composition tools. In that case, a resulting composite web service is seen as a web service in its own right, advertised in the same manner as others in UDDI repositories.

On the other hand, composition can be a dynamic process, whereby an abstract user-formulated goal can be translated on the fly by a background composition program. Such an approach requires the capacity of fully handling the logical conditions that govern the possibility of retrieving/combining web services, and to solve the associated planning problem: at the start we know the goal and the available user inputs, and we expect as a result a complete choreography of web services having the desired result. For practical reasons, it can be expected that dedicated generic web services or web sites will be responsible for interacting with the end user in that case.

The design case may fully benefit from the technical solutions available to solve the automatic composition problem.

4.1.1 Requirements

Composition requires workable results concerning:

- The WSMO model for capabilities
- The WSMO model for interfaces: choreographies and orchestrations
- The WSMO model for goals
- The framework and architecture for SWS discovery

4.1.2 State of the Art

Composition of web services currently uses standards such as Web Services Business Process Execution Language (WS-BPEL) [4] or the Web Service Choreography Description Language (WS-CDL) [1] discussed in detail in Section 2. State-of-the-Art of composition is presented in more detail in [16].
4.2 Discovery

Discovery deals with the problem of finding web services, among a set of advertised web services, that provide the functionality requested by a human or automated software agent. This process is often termed *matchmaking* and may come in two flavors: The agent may either manually browse through the set of advertised services until it has found an appropriate match. Or it may submit a *query* that will be used to filter out services that are not important. This query is formulated as a set of criteria that need to be matched, which, depending on the capabilities of the discovery component, may range from simple syntactic keyword matches up to abstract and complex goal descriptions using a set of ontologies. Although DIP does not exclude the simple case, its main focus will be on semantic concepts and technologies that can be used to match web service capabilities.

### 4.2.1 Requirements

The SWWS report [36] describes a number of fundamental tasks, which lead to the following requirements:

- **A Service Modeling Ontology** is needed to capture the semantics of web services. It can be used to formally describe the capabilities of web services which can then be matched with the goals that an agent may have.

- Such formal service descriptions will be stored in a **Service Registry**. Ideally, the discovery component will be decoupled from the registry component allowing different registry implementations.

- Query-based discovery requires an appropriate **Query Language**.

Based on the work done in [49] we can further refine the requirements for the service description and query language:

- **High degree of flexibility and expressiveness including semi-structured data**: Different service providers will want to describe their services with different degrees of complexity and completeness.

- **Support for types and subsumption**: Matching should not be restricted to simple string-matching. Subsumption can be used to support more complex queries where the relationships between the types to return better matches.

- **Ability to express restrictions and constraints**: Service descriptions or queries often refer to conceptual definitions of suitable services rather than individual instances. This is normally done using restrictions and/or constraints.

Given the fact that the discovery component is supposed to be decoupled from the registry, one can easily think of scenarios where it is desirable to incorporate existing infrastructures, like, e.g., UDDI repositories. In that case, it should be possible to augment existing web service descriptions with semantic information with minimal redundancy.

Last but not least, discovery should be available both at design time and at run time: During the design of a composite web service matching a higher-level goal, the service composer may use the discovery component in her/his development environment to locate matching services for the individual sub-tasks described by proper sub-goals. Using the matches found she/he then links in those web services to eventually build up...
the orchestration of the composite web service. If any of the sub-goals cannot be fulfilled by a matching web service at design time, an agent-based execution environment may then use the discovery service to find appropriate matches at runtime.

4.2.2 State of the Art

UDDI has become the de-facto standard to provide a general framework to describe and discover web services and web service providers. More specifically, [9] is a technical note that details how WSDL descriptions of web services can be mapped to UDDI data structures and provides examples of how one could find web service descriptions using the standard UDDI query interface.

Within the academic world, a number of approaches exist that try to build semantically enhanced discovery components on top of UDDI: [29] and [2] augment the standard UDDI registry APIs with semantic annotations (see also Section 2.6.1). [51] uses a set of distributed UDDI registries as a storage layer, where each registry is mapped to a specific domain based on a registry ontology.

A very active field in research is the development of discovery algorithms. The main focus is on finding good “approximations” when a perfect match cannot be found, i.e., the advertised capabilities of a web service do not fully match the request. [43] defines an ordered scoring function based on input and output parameters as follows:

- **An exact match** is found if advertised and requested inputs/outputs fully match.
- **A plug-in match** occurs if either the advertised outputs or the requested inputs are more specific than their corresponding counterparts.
- In a **subsumed match** the requested outputs or the advertised inputs are more specific than their corresponding counterparts.
- **A failure** occurs if none of the above matches succeeds.

A similar approach is taken in [33] and applied to concept-based reasoning in Description Logics: Here, the above categorization is extended by an *intersection match*, where the intersection of the concepts of the request and the concepts of the advertisement is satisfiable. This intersection match is ranked below a subsumed match.

[6] describes an interesting different approach where discovery is treated as a query rewriting problem on hypergraphs: It attempts to find the so-called *best profile cover* which is defined as the set of web services that satisfies as much as possible the outputs of the request while requiring as few as possible non-requested inputs.
4.3 Invocation

Invocation is the task of performing an actual call to an instance of a web service, i.e., providing it with the required inputs, whereby the service will carry out its function and return any outputs.

This is normally realized by an exchange of SOAP messages. A request message is sent to the web service providing the inputs and any other information necessary. The service then performs its function and returns a response message containing the outputs of the service. More complex scenarios are possible, such as one using callbacks, for example.

4.3.1 Requirements

It is commonly agreed that the invocation component belongs to the implementation level. The invocation component is indeed only a marginal component of the DIP architecture, and does not play such a prominent role as, for example, discovery, composition or mediation. As a matter of fact, the invocation component does not deal with any semantic information at all. It simply performs the mundane task of placing a call to a remote web service according to a description and a payload that it receives from client components.

We expect that the description of which web service to call and how to call it will be described in a WSDL document that is part of the inputs of the invocation component. Other inputs include the actual payload of the request (i.e., the data passed as a parameter of the call to the remote web service).

The invocation component would need a defined API to allow other components to use it. It seems natural to want to express the functionalities of the invocation component as a web service. However, this solution introduces an obvious chicken-and-egg problem: other components in the DIP architecture would need to use the invocation component in order to call the invocation component. A possible solution to this conundrum would be to provide a small library that client components can use to call the invocation component.

4.3.2 State of the Art

Web services are usually invoked by sending a SOAP message to a remote HTTP server using an HTTP GET or POST request, to which the service responds using the same protocol. Details of the message format (e.g., XSD types) and URI of the service are described in a WSDL file.

Current semantic approaches to describing web services (e.g., DAML-S, OWL-S) provide a mapping of the semantically described service to a concrete instance of the web service (known as the grounding) described in WSDL. The SWS browser tool developed by BT is capable of invoking web services described in OWL-S and DAML-S by extracting the grounding information and invoking the service directly via SOAP/HTTP after prompting the user for inputs. We believe that the task of going from a semantically-enriched description of a web service to a semantics-free description (i.e., a regular WSDL document) happens outside of the scope of the invocation component.

In the BT SWS browser [13] composed services are invoked by invoking the atomic sub-services, caching the output and passing it to the next service in the composition. At
present there is no support for service substitution or fault handling, so if one atomic service fails to be invoked the whole composition will fail. Again, our view on the subject is that the execution of a composition of services is not part of the invocation module.

There are already a number of mature tools and APIs for invoking web services:

- The Apache WSIF framework is part of the Apache project and is a set of APIs for dynamically invoking web service based on their WSDL description (http://ws.apache.org/wsif/).

- Web methods GLUE (http://www.webmethods.com/solutions/wM_Glue/) is a Java API that provides support for XML, SOAP, WSDL and UDDI. This API is a good basis for developing tools to invoke web services and is used by the BT SWS browser.

- The BEA web logic Server is a Java development platform that allows a programmer to automatically create web services from Java classes (http://dev2dev.bea.com/products/wlserver81/web_services.jsp). It automatically generates the WSDL description, hosts the service and provides an HTML form to invoke the service via HTTP. In our view, the generation of the implementation of the service is outside of the scope of DIP. The generation of a web form that a human could use for invoking a web service is interesting but peripheral to our concerns in DIP, which is the automated invocation of web services.
4.4 Mediation

DIP does not assume a homogeneous world and tries to support interoperability across heterogeneous systems through the concept of **mediation** as introduced in [56]. In [26] mediation scenarios are identified at various levels:

- **Data mediation** helps to link services by transforming different input/output structures and/or different data types for equivalent input/output messages and their elements.
- **Protocol mediation** deals with mismatches in message exchange patterns.
- **Process mediation** supports interoperability between services where the process models, like sequence of activities, states and state transitions, and control and data flows differ.

DIP will primarily focus on data and process level mediation.

Traditionally, transformation is applied on a pure syntactical level, e.g., using XSLT to map between XML-based data representations. In contrast to that, **semantic transformation** as envisioned in DIP tries to provide mapping schemes based on semantic concepts, typically described in domain-specific **ontologies**. **Business Data Ontologies** will help to mediate between heterogeneous business data models, whereas **Business Process Ontologies** will try to link between different business processes.

### 4.4.1 Requirements

Although the actual mediation tasks outlined above will happen at runtime, we do not expect that the corresponding mapping schemes/rules can be derived automatically without human intervention (automated approaches do not meet industry-strength requirements at the moment). Thus the overall mediation component should be divided into

- a **run-time environment** that will carry out the transformations,
- a **design time tool** that will assist humans to define the transformations rules that are applied at run-time,
- a **formalism (algebra)** to specify the mapping rules, and
- a **representation scheme (language)** to describe the instances to be mapped.

DIP cannot and will not provide a full solution right from the beginning, but will rather start with a simple set of capabilities that will eventually evolve into a full solution. Furthermore, given the limited time and resources, the project will not be able to start from scratch, but will probably make use of existing environments and infrastructures. Taking these requirements one step further, the most appropriate approach to accomplish this goal is to decouple the various components in the overall DIP architecture. For example, the mediation component should not be locked into a particular web service execution environment, thus allowing partners to experiment with and extend different approaches.

This decoupling will probably be accomplished by turning the mediation run-time component itself into a (set of) web services that can be discovered and used by other system components. This discovery may again be based on some type of ranking (see Section 4.2), i.e., a mediation service description may include (non-functional)
properties that will tell the requester how “well” the service can perform a particular transformation.

At best, any development within DIP can be deployed in existing infrastructures adding functionality “on top” while minimizing the need to replace components that are already in place. From this, one can conclude that both the service requester and the service provider are totally unaware of the fact that mediation takes place when talking to each other.

This transparency, though desirable, points to a number of new problems that need to be addressed in the overall architecture: Any mediation step that is performed on the way from the requestor to the provider needs to be logged for auditing purposes. This is especially crucial in use case scenarios where web service invocations may be part of legally bounded interactions (cf. 3.2.3). As a consequence, mediation will have an impact on security and trust: Since it will potentially change data, protocol and/or process elements, and parties with high security requirements will only be willing to include mediation services they trust and that will guarantee data/process integrity and confidentiality.

4.4.2 State of the Art

Data level mediation in existing commercial products is primarily based on syntactical transformations, where the transformation rules are provided by a human. Typically, rules can be created either with a graphical tool by connecting corresponding message elements or by programming them in a (tool-specific) scripting language. Some tools also provide a mixture of both or even allow the user to extend the mapping language with user-specific functions. When it comes to execution, the run-time environment either interprets the rules, or the mapping is executed by calling compiled code that represents the rules. In research, a number of different approaches have been proposed to address the data mediation problem at a schema level, which are surveyed in [46]. The most advanced system is COMA [16], which implements a very flexible mediation approach: The matching process actually consists of one or more iterations of a three-step pipeline. The first phase is an optional user feedback phase on the matching results; then, the individual matchers taken from a matcher library will be executed, which, in a final phase, will be combined into the overall matching result. The system is easily extensible by adding new matchers to the library.

At present we do not know of any system that addresses the process mediation problem as outlined above. This field represents an area of ongoing research where existing approaches are currently focusing on logically formalizing process choreographies [9] as well as dynamic aspects of such composite processes [24].
4.5 Ontology

Ontologies are the primary means for enabling semantic integration of data and processes in heterogeneous distributed systems. An ontology is a formal conceptualization of some problem domain that represents a shared and common understanding of that domain by a group of interacting agents. Thus, ontologies represent knowledge about an underlying problem domain that is needed for bridging the gap between the “languages” and “perspectives on the common world” which are used by the single interacting agents. By using the vocabulary defined in an ontology for constructing messages, exchanged data can be semantically interpreted by the receiver and seamlessly integrated in the particular data schema. In this way, ontologies allow agents that use different data schemas to understand the messages received during a conversation on a semantic basis.

Ontologies are the key component for the semantic web and semantic web services and hence are used by all components of the DIP architecture in order to retrieve ontologies, concepts, and their instances and access the knowledge that is formally represented by an ontology for facilitating interoperability between web services.

Within DIP the component “Ontology” has the following objectives: provide a highly functional authoring and management component supporting editing, browsing, validation, and consistency of ontologies. This means especially, that the component includes a semantic query engine (along with a query language) for querying ontologies. The query engine enables clients to exploit the knowledge contained in ontologies for integrating interacting agents, in particular web services.

A collaborative authoring environment should be available for building complex ontologies to enable several designers to cooperate in building an ontology concurrently. In particular, this means that we have to cope with heterogeneous networks of “simple” ontologies that have to be combined into complex ontology networks. Therefore, the integration has to deal with conflicts and contradictions between the single ontologies in the network. Since ontologies represent an agreed view on (a part of) the world at a certain point in time and because the world underlies continuous changes, ontologies, are quite naturally subject to changes too. Hence, the component should allow clients to create, modify, and store multiple versions of ontologies. The component has to support the concurrent management of multiple versions of many ontologies.

4.5.1 Requirements

As elaborated above, the ontology component is a basic one, which is been used by the rest of the components without considerable dependencies in the opposite direction. Still, it has to be mentioned that the web-service description framework (workpackage 3) and the overall architecture should allow the ontology component to be described and used as a provider of ontology management (OM) services – storage, retrieval, querying, alignment, etc. The specific requirements of this type of services are related to the fact that they manage or offer information. Because of this, they have an impact on the information space, but (typically) not to the real world.

These requirements of the ontology component (taken as OM service provider) however are quite similar to those coming from some of the services considered in the case studies. It is obvious, that the above “contradiction” could easily be solved by more
careful definition and elaboration of the scope of usage of the assumptions and effects in WSMO and, more generally, some comments on what can be considered a change in the real world.

Further, there are a number of basic requirements which the overall architecture and the case studies put towards the design of the interfaces of the Ontology component:

- **The Ontology component should be configurable** – there could not be a single piece of software performing all the OM-related functionality for all the possible uses. The ontology component will often be a combination of different modules, e.g., specific repository, reasoner, ontology mediation service.

- **It should not be bound to a particular ontology language.** This is due to a number of reasons (see [47] and [18]), but most essentially (i) different ontology languages are suitable for different purposes and (ii) there are no well-established, scalable and reliable, ontology reasoning and querying tools.

- **It should allow independent OM tools development** (to facilitate different configurations). Development of ontology management infrastructure and tools (e.g., for persistence, visualization, evolution, and mediation) which are as much as possible independent of the choice of formal semantics, reasoning strategy, and implementation.

- **Reuse of OM servers and tools** in a language-independent manner. In many cases and for many of their functions, the OM tools are not strictly dependent on the semantics of the knowledge being managed. For instance, it is natural for a knowledge access tool to be used with both F-Logic and OWL reasoners.

There are number of general requirements towards the interfaces of the ontology component:

- **Support integration of multiple complementary OM Servers.** For instance, it should be possible that both a reasoner and an ontology repository implement interfaces and collaborate using them.

- **Enable an OM system to mediate** in front of another OM system. There could be variants of mediation between two OM systems or between an OM system and a client. Multiple scenarios have to be possible (a more detailed analysis is provided in [21]):
  - To mediate representation or behavior differences, between systems using different ontologies, representation languages, or protocols.
  - A mediator could be used to enforce an access control policy between the client and the OM system in order to (i) ban calls which are not allowed to the client and (ii) filter results.

- **Support wrapping of non-ontological structured data-sources.** It should be possible that a DBMS is being wrapped as an OM system in order to provide access to all or parts of its data (obviously, without inference).

- **Dataset and ontology modularity.** Dataset modularity (which includes ontology modularity, as far as the latter is a special sort of dataset) is important for the following purposes:
  - The data-sources should be able to determine the scope of the information they support/expose/offer.
It should be possible to specify the complexity of (the expressivity used in/necessary for) particular datasets;

It should be possible to define dependencies between datasets (ontologies or instance data);

It should be possible to define distinct access rights to the different datasets or parts of them;

It should be possible to assign non-functional properties to the datasets (for support, maintenance, etc.)

An extended analysis of the requirements for the OM infrastructure is provided in [17]. Further, an extensive discussion on the structure of the ontology management tasks and problems is provided in the beginning of [18].

Discussion on the requirements towards the ontology and knowledge representation languages is presented in [16]. A draft of a similar comparison, however, with a different focus of coverage and methodology, is presented in the WSML deliverable D8 Language Comparison and Evaluation [15]. Finally, a definition of language-neutral epistemological ontology model designed to meet the requirements for usage of ontologies within WSMO is provided in D18 of WSML [47].

4.5.2 State of the Art

State-of-the-art is covered in detail in [16] and [17] of the DIP project. Thus, we only give a brief summary here.

Regarding ontology reasoning and querying, the most popular standard is OWL. Description Logic provides proficient facilities for reasoning about ontological structures, thus OWL-based reasoning mechanisms support this. But, due to the nature of the description language, these mechanisms do not provide sufficient support for reasoning about instance data – which is an essential requirement for sophisticated ontology querying systems. Besides, reasoning support for dynamic information structures, as a new asset arising within semantic web services, is not supported by currently existing ontology reasoning and querying techniques.

Regarding ontology management, currently existing techniques provide separated solutions for editing, storage and retrieval, versioning and evolution of small-sized, homogeneous ontologies that are not interrelated. Although there are promising approaches and technologies existing at this point of time, there exists no platform that provides a high-quality integrated ontology management environment.
4.6 Publishing

Publishing a DIP element (web service, ontologies, or goals) means to make information about a DIP element available so it can be retrieved using certain search criteria. DIP will use a registry to publish and retrieve this information.

DIP elements are published by providing only the necessary information to the registry that is needed for the syntactical and structural retrieval as well as the links to the detailed information about DIP elements. Some of the semantic description elements have to be mapped and converted to be used for structural retrieval. The registry also must maintain the publishers and relate the stored information to them.

Publishing a semantic web service can involve one or more of the following aspects:

- Generating a formal SWS interface definition from a specification (e.g., UML) or from existing code (e.g., Java/EJB)
- Adding a (formal) description of the service to a registry
- Further describing a service by adding metadata and/or informal descriptions not contained in the formal service description

4.6.1 Requirements

- Formal description of the semantic web service (SWS’s equivalent of WSDL)
- Address of the registry (if not specified, a default registry could be used)
- Publishing uses the registry component for persistent storage of service descriptions.
- A formal service description language for semantic web services
- The API of the registry
- Optional: additional information about the service – e.g., non-functional properties

Instead of WSDL a semantically enhanced interface description language will be used in DIP (e.g., based on WSMO). To guarantee compatibility with standard web services and to leverage existing tools this enhanced interface language should be somehow mapped to WSDL.

Although there might be conceptual objections against the “bottom-up” approach to generate WSDL from code, one should keep in mind that this is common practice in development projects. It allows developers to continue working in their preferred environments and makes web services an easy to add, evolutionary extension. One challenge here would be to support developers in generating semantic web service from code.

Another direction would be to start with UML specifications and use them as a basis for publishing semantic web services.

The major challenge will be to formalize semantic aspects of service descriptions and to integrate them into the service description language instead of having them as informal natural language texts in some registry fields.
4.6.2 State of the Art

Publishing of web services currently provides the following functionality:

- Generation of WSDL from existing code (e.g., Java/EJB). This is a standard functionality of development environments (e.g., Visual Studio .NET), open source web service tools (e.g., Axis) and application servers (e.g. IBM Websphere, BEA Weblogic).
- Registering a web service in a UDDI registry by using a UDDI API.
4.7 Registry

DIP introduces goal-driven processing, which increases the importance of discovery, and therefore potentially the importance of registries.

Goals decouple requester and provider. They separate requester and provider at least by function (goals may specify functionality which may even not exist at the time the goal is created), time (the service to satisfy a goal may be found and invoked much later than the goal is created), and by resource (the requester may not know in advance who will finally resolve the goal).

Whatever discovery mechanism is used, it must finally “undo” the separation, i.e., link together goal, service and resource, and let the service be invoked. This needs support through ontologies and other semantic processing technologies such as reasoning and mediation.

Currently, UDDI and related concepts only provide very limited semantic support, if at all which limited the possibilities of goal-driven processing.

4.7.1 Requirements
- Define a publish/retrieval procedure for web services, goals and ontologies (DIP elements)
- Reuse and adapt existing and well accepted standard methods and tools instead of specifying a new registry from scratch
- Provide the possibility to extend existing web services with DIP specific element properties (non functional properties, semantic description)
  - Allow coexistence of existing web services and SWS
  - Store and maintain information on web service publishers

4.7.2 State of the Art

UDDI [50] is currently the de-facto standard for web service registries with current attempts to enrich it with semantic information. UDDI and its proposed extensions are discussed in detail in Section 2.6. An approach to extend UDDI for semantic web services used by the DIP project is described in [20].
4.8 SWS Description

The objective of this component is to support easy definition, description and publication of semantic-based services. The research issues to be tackled are the following:

- Enhanced syntactic and semantic definition and discovery of SWS.
- Decentralized registries that enable automatic registration of SWS at the place they are offered.
- Semantic query languages to discover SWS.
- Automatic registration and search of SWS.
- Overcoming the “hard-wiring” problem of service composition.
- Quality of service descriptions do not support composite web services and the proposals are not aligned with metadata proposals for the Semantic Web.

4.8.1 Requirements

- provide support for domain knowledge mediation at discovery time
- explicitly support semi-automated composition techniques

4.8.2 State of the Art

Currently, web service definitions and descriptions are restricted to the interface level on the basis of syntactic descriptions. Publication is done by registering services in a centralized repository such as UDDI repository. UDDI includes a very flat metadata model to describe services in terms of yellow, white and green pages. Besides UDDI, web services can also be published using WS-Inspection [55] that allows publishing of a service on an arbitrary web server. Again, the WS-Inspection metadata model is extremely simple. An important issue in this context for this component is the quality-of-service description. Only first proposals on policy descriptions have been made.
5 NON-FUNCTIONAL REQUIREMENTS

This section defines the non-functional requirements of DIP. The non-functional requirements described in [9] are included and not explicitly listed again.

5.1 Trust

A main goal of the web services is to build an infrastructure, which allows applications to use services provided by large networks of independent service providers. Using services from different sources automatically raises questions about service quality and trustworthiness. Before the service can be used, its quality and trustworthiness have to be evaluated according to application-specific trust requirements. This trust decision can be based on one or more of the following trust mechanisms:

1. **Reputation-based trust mechanisms** include web-of-trust and rating systems like the one used by eBay. There has been a lot of research about different trust metrics for reputation-based trust mechanisms. The general problem of all these approaches is that they require explicit and topic-specific trust ratings and that defining such ratings and keeping them up-to-date places a big burden on information consumers.

2. **Context-based trust mechanisms** use meta-information about the circumstances in which information has been claimed, e.g., who said what, when and why. They include role-based trust mechanisms, using the author's role or his membership of a specific group for trust decisions. Example policies are: "Prefer product descriptions published by the manufacturer to descriptions published by a vendor" or "Distrust everything a vendor says about its competitor."

3. **Content-based trust mechanisms**: These approaches do not use metadata but the content and related information for trust decisions. New information is compared with information published by other authors. An example policy would be: "Believe information which has been stated by at least 5 independent sources." Another approach is to compare new information with information or rules believed to be true, for example, "Distrust product prices that are more than 50% below the average price."

Also combinations of these policies are possible and make sense. For example, a policy that combines reputation-based trust mechanisms (ratings) with role-based trust mechanisms could state "Retrieve all persons with the skill 'java programming'. Rank the results according to the number of positive ratings they received from other programmers." The policy uses only ratings from other programmers and would skip ratings from project managers.

5.1.1 Requirements

[8] proposes that a trust architecture for addressing the trust scenarios of the Semantic Web, should meet the following design principles:

- **Decentralization**: Truth - or more pragmatically trustworthiness - should be evaluated on a subjective basis by each application that processes information found on the web. The architecture should not require central trusted third
parties and should not require users to follow a single schema for trust assertions.

- **Tolerance of incomplete information**: The architecture should impose minimal or no requirements on information providers. The system should not exclude information providers who have not been rated or do not publish trust relevant information according to a specific schema. On the other hand, the system should be able to use all trust relevant information (provenance, context information and ratings) published or generated in the information gathering process (source URL, date).

- **Support for different trust policies**: There will not be a single best trust metric for all situations. Thus the architecture should support a mixture of different mechanisms depending on the level of trust required by a specific application and the context and background information available. The architecture should support users to formulate subjective and task-specific trust policies.

- **Ranking of information**: Instead of eliminating distrusted information the architecture should use a more “google-like” approach and rank information according to its trustworthiness.

- **Justification of trust decisions**: There is no absolute truth. Seen from different views of the world, contradictory assertions can be simultaneously true. As a consequence a trust architecture should be able to explain which statements have been used for trust decisions.

These requirements clearly go beyond what is usually handled by the trust approaches known from networking (public key infrastructures using trusted third parties) and agent based systems (agents are usually required to register with a specific agent environment and use the trust metric provided by the environment). As trust is important to web services but not at the core of the DIP project we intend to reuse and apply exiting infrastructures from industry, other EU projects and academia.

**5.1.2 State of the Art**

Nowadays, online communities (slashdot.org, affero.org), e-marketplaces (ebay.com, amazon.com, epinions.com, etc..) and peer-to-peer networks (edenkey, Gnutella) use and make trust information available (judgment expressed by users on users). Most of these systems store the trust ratings of communicating partners in a centralized location and thereby violate already the first requirement defined above.

The parties in peer-to-peer networks are an exception since there does not exist a central store for information. The last generation of peer-to-peer networks brought up structured systems such as P-Grid [43]. [1] gives an example of how trust can be managed in decentralized environment in a robust way. The basic idea is that the value of trust for a certain party is stored in a decentralized manner, like any other object in the system.
5.2 Security

Currently, secured web services are used internally in organizations or in B2B environments to streamline business processes or in B2C environments (for example, online payment with credit cards). In some of these cases, ensuring simple web service security can be accomplished by using currently available techniques. Depending on the deployment model, lower network layers can provide some basic security services for web services. For example, point-to-point interaction in a B2B environment can be authenticated and secured using layer 3 VPN technology or SSL/TLS.

In the general case, this simple approach will not be sufficient. With the introduction of routers into the web services world the need for end-to-end security that persists beyond the transport layer emerged. But it must still be possible for the router to read or even modify some of the information transported in the SOAP message. With the introduction of mediation and composition this situation is even further complicated: It may be that an intermediary party has to be able to access the transported data. Is it possible to keep up end-to-end security in this case?

In order to ensure the further success of semantic web services in B2B and B2C environments, a number of on-going efforts aim at developing a flexible and expandable security architecture for the current generation of web services. The security of web services should not be left in the hands of developers that are generally more concerned with meeting deadlines and typically are not security experts. This motivates the need to include structured and comprehensive security mechanisms in the semantic web services architecture.

5.2.1 Requirements

In [5] the requirements for providing end-to-end security for web services are specified as follows:

1. Authentication mechanisms: This is needed in order to facilitate mutual authentication of service provider and service invoker.
2. Authorization to access resources: Once authenticated, authorization mechanisms control invoker access to appropriate system resources. There should be controlled access to systems and their components.
3. Data integrity and confidentiality to ensure that information has not been modified during transmission and is only accessible to intended parties. Encryption technology and digital signature techniques can be used for this purpose.
4. Integrity of transactions and communications: This is needed to ensure that the business process was done properly and the flow of operations was executed in a correct manner.
5. Non-repudiation so that a participant of a transaction cannot deny the occurrence of the transaction.
6. End-to-end integrity and confidentiality of messages: The integrity and confidentiality of messages must be ensured even in the presence of intermediaries.
7. Security and audit trails: This is required in order to trace user access, behavior and enable system integrity verification.

8. Distributed enforcement of security policies: Implementers must be able to define a security policy and enforce it across various platforms with varying privileges.

All the mentioned security requirements must work with the general web services framework including WSDL service descriptions, UDDI service registrations and the SOAP message structure and message processing model.

As security is important to web services but not at the core of the DIP project we intend to reuse and apply existing infrastructures from industry, other EU projects and academia.

Availability, trust management and application level security are not part of this component.

5.2.2 State of the Art

In [27] IBM, Microsoft and other major companies have defined a web services security model that guarantees end-to-end communication security. Figure 5 illustrates the building blocks and the dependencies of the model.

![Figure 5: IBM/Microsoft web services security model](image)

The architecture uses the existing W3C XML standard recommendations XML-Encryption [61] and XML-Signature [68] as a basis for a family of web service standards.
5.2.2.1 XML Security Base

The XML-Encryption standard proposal specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

The XML-Signature standard proposal specifies XML syntax and processing rules for creating and representing digital signatures. XML Signatures can be applied to any digital content (data object), including XML. An XML Signature may be applied to the content of one or more resources. Enveloped signatures or enveloping signatures over data are within the same XML document as the signature; detached signatures over data are external to the signature element.

Both standard proposals are based on the W3C note XML Key Management Specification [67] of protocols for distributing and registering public keys.

5.2.2.2 The IBM/Microsoft WS-* series of proposals

The base component of the standard proposal by IBM and Microsoft is the WS-Security specification [61]. It is an OASIS standard since May 2004 and proposes a standard set of SOAP extensions that can be used when building secure web services to implement integrity and confidentiality. WS-Security provides three main mechanisms: security token propagation, message integrity and message confidentiality.

The message authentication mechanism provided by WS-Security is useful for simple or one-way messages. Parties that wish to exchange multiple messages typically establish a secure security context in which to exchange multiple messages. In WS-SecureConversation [62] a security context is shared among the communicating parties for the lifetime of a communication association.

In order to secure a communication between two parties, the two parties must exchange security credentials (either directly or indirectly). However, each party needs to determine if it can "trust" the asserted credentials of the other party. The WS-Trust specification [63] defines extensions to WS-Security that provide methods for issuing and exchanging security tokens and ways to establish and access the presence of trust relationships.

WS-Policy [64] defines a policy to be a collection of one or more policy assertions. However, WS-Policy falls short in specifying how policies are discovered or attached to a web service. This seems to have been done intentionally and this issue should be addressed in the WS-PolicyAttachment specification.

WS-Authorization [27] will define how web services manage authorization data and policies. There is currently no document about WS-Authorization publicly available.

In a trusted domain the source and target of a request can determine and agree whether particular sets of credentials from a source satisfy the relevant security policies of the target. The WS-Federation specification [65] defines mechanisms to allow different security realms to federate using different or like mechanisms by allowing and brokering trust of identities, attributes and authentication between participating web services.
5.2.2.3 SAML *(Security Assertion Markup Language)*

SAML (Security Assertions Markup Language) [41] is an industry standard ratified by OASIS (Organization for Advancement Structured Information Sciences). This XML-based framework provides a standard way to define user authentication, entitlements and attribute information in XML documents.

As its name suggests, SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject to other entities, which may be a partner company, another enterprise application, etc. These assertions are passed as XML documents, either pushed from the Asserting Party to the Relying Party or pulled from the Asserting Party to the Relying Party.

For both B2B and B2C, a single transaction can be distributed across multiple companies, multiple web sites, and multiple marketplaces, all of which may have their own authentication and authorization schemes. Companies need a standard, open framework that will enable them to build trust chains across company boundaries, heterogeneous platforms, and multiple vendor solutions.

5.2.2.4 XACML *(eXtensible Access Control Markup Language)*

XACML (eXtensible Access Control Markup Language) [38] is an XML-based language for access control that has been standardized in OASIS. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (who can do what and when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses).

In a typical XACML usage scenario, a subject (e.g., human user, workstation) wants to take some action on a particular resource. The subject submits its query to the entity protecting the resource (e.g. the filesystem, a web server, etc.). This entity is called a Policy Enforcement Point (PEP). The PEP forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information. The PEP then sends this request to a Policy Decision Point (PDP) which examines the request, retrieves policies (written in the XACML policy language) that are applicable to this request, and determines whether access should be granted according to the XACML rules for evaluating policies. That answer (expressed in the XACML response language) is returned to the PEP, which can then allow or deny access to the requester.
5.3 Transition path to semantic web services

A key issue in the introduction of semantic web services in the industry is the protection of investments in existing web service infrastructures. From a technical viewpoint this means that smooth and incremental transition strategies from web services to semantic web services have to be defined. As this requires a major analysis effort and since semantic web services are currently under development but far from being stable, we can only highlight some issues that appear relevant in this respect. In the course of the DIP project and with more consolidated semantic web service standards a more structured and comprehensive discussion of the issues involved in the transition from web services to semantic web services may be possible.

Semantic web services extend the web service paradigm by adding ontologies (as a carrier of consensual, formal, machine interpretable semantics) in several ways:

- **Ontologies used for WS description** and next ontology-based infrastructure (reasoners, query engines, repositories) used for a more flexible and intelligent web service discovery (based on the semantic descriptions).

- **Ontologies used to facilitate mediation** (of various sorts) to take place at the stage of WS usage.

Two sorts of changes in the non-functional requirements can be considered in the process of transition from WS to SWS:

- Additional non-functional requirements for the ontologies and their supporting infrastructure;

- Changes to the non-functional requirements of the WS, due to the usage of ontologies.

5.3.1 Non-Functional Requirements related to the usage of ontologies

The behavior and specifics of the ontologies with respect to their social aspects (like trustworthiness), maintenance, management, and access control are quite similar to those of the database schemata. Both determine the structure, semantics, and consistency of much bigger amounts of structured data, thus become critical resources. A minor change in an ontology can invalidate, make inconsistent or misleading or useless thousands of SWS descriptions based on it.

There are several non-functional requirements towards the ontologies and the related infrastructure:

- **Authorship and status-related information**, necessary to support the ontology development process. Those issues are the same as for other information resources and a standard metadata set such as Dublin Core seems to be a good fit (see [8]). A general discussion on the methodology related to development of ontologies is given in [47].

- **Security related information.** Here the access to the ontologies (and the data aligned to and accessible through them) has to be defined with respect to there semantics. This could mean, for instance, definition of security policies such as “write permission for X type of attributes of Y class of documents which have relation Z to organization W”. Such security policies require query-like
declarative data-descriptors, based again on the ontologies. A sample schema for this approach is presented in [30].

- **Evolution related information**, including formal handling of the levels of backward compatibility, with respect to the semantics.

### 5.3.2 Changes of the non-functional requirements of the web services

There are no major changes in the non-functional requirements of the web services themselves, related to the fact that they are semantically described and discovered. The requirements for trust and security remain the same disregarding the usage of ontologies, and considering the non-functional requirements about the ontologies (see the previous sub-section).
6 INTEROPERABILITY

This section addresses the issue of interoperability in DIP. It has already been agreed that DIP will use the basic web service standards WSDL and SOAP. WSDL will be used to describe interfaces of DIP components and SOAP messages will be used to access them. Although this is a good basis for facilitating integration of different DIP components, it does not eliminate interoperability problems completely.

In principle, two interoperability scenarios can be distinguished:

1. Interoperability within DIP, i.e. interoperability between components developed in DIP which are in a consumer/provider relationship. Here interoperability can be enforced through DIP recommendations.

2. Interoperability with services developed outside of DIP. Although interoperability cannot be enforced in this case, it is recommended to limit interactions to external services meeting the basic interoperability requirements (as stated below).

In the following we will briefly sketch where interoperability problems could arise and how this can be avoided. It is strongly recommended to base work in DIP on the profiles developed by the WS-I (Web Services Interoperability Organization). WS-I has recently released WS-I Basic Profile 1.0 which should become made a mandatory requirement for all DIP components supporting WSDL and SOAP. Moreover, a tool to be used to test WS-I Basic Profile 1.0 conformance of components should be selected. Interoperability recommendations should be updated periodically as it is expected that WS-I will release new or updated profiles in the course of the DIP project.

6.1 Interoperability Problems in Web Services

Although basic web service standards like SOAP and WSDL are stable and mature, interoperability problems still occur in practice. These problems are mainly due to the fact that existing tools and products only support parts of the standards. And different tools might support different parts of the standards, thereby failing to be fully interoperable.

Let us first consider the usage of SOAP and WSDL in more detail. WSDL is used to describe service interfaces and SOAP is then used to actually access these interfaces. Products offered by software vendors and open source tools provide functionality to automate the generation of both WSDL descriptions and code for SOAP handling.

Most industrial software development environments now offer web service support. This usually means that developers have tools for modeling service interfaces, e.g., graphical interfaces using UML-like notations or editors for specific programming languages such as Java, VB, C++, or C#. WSDL descriptions are then generated automatically in a second step.

The WSDL interface specifications can then be used to automatically generate code connecting the corresponding SOAP messages to a specific programming language. On the client side, such “glue” code offers the appropriate programming language construct, e.g., a Java method, which can be called by application programmers. When the method is called, it transforms the arguments into XML data and generates the corresponding SOAP call to a service operation. It then receives the SOAP response
message, transforms the results from XML into programming language constructs and returns the data to the caller of the method. On the server side, the glue code performs the reverse transformations.

One major benefit of web services are their independence of programming languages, platforms and vendors. In an ideal world one should be able to produce a service interface in WSDL which system A and system B should be able to read in and automatically generate glue code for SOAP. Service calls from clients implemented with system B should then interoperate seamlessly with services implemented with system A.

In practice, however, this interoperability does not always work flawlessly. Problems of interoperability arise from the standards involved, namely XML Schemas, SOAP, and WSDL. Though vendors claim that their products support these standards, support is usually limited to a subset. Thus system A may support a specific construct from XML schemas and hence use it when it generates WSDL files. However, system B might not support this specific construct and will thus be unable to process the WSDL file and generate SOAP glue code from it.

There are also cases where the standards are ambiguous and different products interpret the standard differently. Again, this leads to interoperability problems.

It should be noted that such interoperability problems usually are not show stoppers in development projects. In most cases, problem sources can be identified in a reasonable amount of time and then work-arounds and fixes can be installed, avoiding those constructs which cause the interoperability failure. However, a more systematic approach to interoperability is needed to reduce overhead caused by such failures. The WS-I described in the next section has been founded to address this issue.

6.2 WS-I (Web Services Interoperability Organization)

WS-I (Web Services Interoperability Organization) has been established to develop solutions for overcoming interoperability problems of web services.² WS-I is an open, industry organization chartered to promote web services interoperability across platforms, operating systems, and programming languages. Since its formation in February 2002, more than 170 companies have joined WS-I.

6.2.1 WS-I Basic Profile 1.0

In April 2004, WS-I has released WS-I Basic Profile 1.0, consisting of a set of non-proprietary web service specifications, along with clarifications and amendments to those specifications which promote interoperability. This profile contains requirements for messaging (SOAP), service description (WSDL) and registration/discovery (UDDI).³ The corresponding “artifacts” are called MESSAGE, DESCRIPTION, and REGDATA.

In particular, WS-I Basic Profile 1.0 resolves more than 200 interoperability issues associated with using the core web services specifications, including⁴

² http://www.ws-i.org/.
⁴ http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0faq.pdf.
• “RPC-encoded” was deprecated (indicating a shift to the use of schema as the interoperable type system)
• Support and guidance for using rpc/lit, and how to construct rpc/lit messages
• Agreement on common interpretations for various aspects of WSDL (e.g., discrepancies between the specification, schema, examples, and appendix mean that WSDL could be – and has been – interpreted in different ways, which can lead to interoperability problems)
• Unique signatures for input messages
• Clarification of SOAP failures, and development of complete error-handling conventions
• Resolution of issues related to HTTP binding, HTTP status code, and SOAPAction headers
• Clarification about “one-way messages”

The requirements concern

• INSTANCE: a deployed instance of a web service, i.e., software that implements a “wsdl:port” or a “uddi:bindingTemplate”.
• CONSUMER: software that invokes an INSTANCE.
• REGISTRY: a UDDI registry, capable of managing REGDATA.

Furthermore, web services instances and consumers have to satisfy requirements both as SENDER and as RECEIVER of messages.

6.2.2 Applying WS-I Basic Profile

Applying the WS-I Basic Profile in DIP would have the following consequences:

1. All WSDL interfaces provided by DIP components would have to conform to the profile.
2. All SOAP messages exchanged between DIP components would have to conform to the profile.

Note that this is only the minimal consequence of profile conformance. It basically means that no component produces output which does not conform to the profile. However, “real” conformance goes a step further, since it implies that a component adequately handles all input which is conformant.

Most software vendors supporting the WS-I profile provide functionality to automatically test profile conformance. There are also open source tools available offering such automatic conformance testing, e.g. the tools provided by WS-I itself (C# and Java)\(^5\) or Axis.\(^6\)

---


\(^6\) http://ws.apache.org/axis/.
6.3 Interoperability Recommendation for DIP

Based on the above presentation the following recommendations are proposed to facilitate interoperability in DIP:

1. It is recommended that all WSDL interfaces provided and all SOAP messages exchanged in DIP are based on WS-I Profile 1.0. Such a requirement should be formally issued by the Technical Project Management Board.

2. The Technical Project Management Board should also select a tool for automatic profile conformance testing of DIP components (e.g., the tools provided by WS-I itself).

3. The Technical Project Management Board should periodically update its recommendation and include new or updated profiles releases by WS-I.

4. In addition DIP components should aim at avoiding the unnecessary use of complex constructs from XML Schemas, SOAP, or WSDL. Early integration tests should ensure that clients and servers actually interoperate on the syntactic level.

5. In addition to the profiles developed by WS-I, some WS standards, such as WS-Policy\(^7\) and WS-MetadataExchange\(^8\) are currently emerging which might help to reduce interoperability problems regarding non-functional properties. It is recommended to closely monitor these standards and to consider their usage in DIP.


7 CONCLUSIONS

In this deliverable we have presented the state of the art in web services architectures and requirements and have discussed their relation and relevance for DIP. As DIP is an effort to evolve current web services architecture into semantic web services we have addressed the novel requirements of DIP beyond the state of the art and suggested strategies as to how they can be put into practice in course of the project. We have focused on the novel requirements induced by the introduction of semantics into all aspects of the architecture. For the “standard” areas we will rely on existing technologies and approaches as much as possible (and extend them where necessary) to enable reuse of existing infrastructures and facilitate smooth transition from non-semantic to semantic web services.

The existing base standards that DIP will rely on are SOAP and WSDL. In the domain of choreography/workflow it is to be decided whether we will base our work on WS-CDL or WS-BPEL. This will have to be determined in the course of the project and depends on a number on factors that can only be clearly determined after more research has been done in the related workpackages. Regarding registries, UDDI is clearly a standard we have to incorporate but it is already clear that we have to considerably extend it to be able to support semantics in the way and to the extent envisioned in DIP. For security related aspects we will clearly rely on the WS-* group of proposals as security is highly relevant but beyond the scope of DIP. Application-domain specific standards such as ebXML will not be used in DIP itself but certainly guide the development process to ensure applicability of the DIP infrastructure in these domains.

A key issue of DIP will be the semantic description of web services to facilitate semantic/goal-oriented discovery and inference mechanisms which will both be needed for various areas in DIP, for example, mediation which will be a key contribution of DIP. In this area few proposals exist which can only be used to a limited degree to achieve DIP’s goals. Thus DIP will have to provide the necessary technologies by itself. A key technology in this area will be the upcoming WSMO standard which is under development in DIP as a project participating in the SDK cluster of EU projects.

As WP6 is concerned with DIP’s overall architecture and interoperability this document can be seen as an umbrella integrating the state-of-the-art and requirements analyses done by the other technical workpackages (WP1–WP5).

This deliverable contributes to the major DIP result of defining an open source SWS architecture and thus has a major impact on achieving the goals of the DIP project.
REFERENCES


World Wide Web Consortium: Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsd1


