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

Deliverable

D4.2
Publishing process specification

Reinhold Herzog
Peter Zugmann
Joachim Quantz

June 22nd, 2004
EXECUTIVE SUMMARY

This deliverable focuses on how DIP Web Services, Mediators, Goals and Ontologies can be published and retrieved using a Registry\(^1\). Publishing information is necessary to enable potential users of the information to retrieve it from a well-known location without needing to know who is actually providing the information.

The approach to Registry proposed in this deliverable aims at ensuring coexistence of DIP Web Services and existing Web Services and providing a way to semantically enhance the latter. It also allows using readily available tools for publishing and thus minimizes the development effort within DIP.

It is proposed to use a UDDI V2 registry as a basis for DIP and to store semantic information at a location chosen by the information provider, i.e. not in the Registry itself, which only contains pointers to the semantic descriptions. This ensures a clear separation between basic Registry and Retrieval functionality and that a more sophisticated semantic search function can be provided by Discovery components.

The deliverable describes in detail how to implement a DIP registry based on existing tools (UDDI) and specifies the necessary programming interfaces. It should thus enable a quick set-up and use of a Registry in DIP.

\(^1\) Disclaimer: DIP Consortium proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
### Document Information

<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 4.2  **Title** Publishing process specification  
**Work package** Number 4  **Title** Service Usage

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual M 6</th>
<th>Actual 22-Jun-04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>version 1.00</td>
<td>final ☑️</td>
</tr>
<tr>
<td>Nature</td>
<td>Prototype ☐</td>
<td>Report ☑️</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public ☐</td>
<td>Consortium ☑️</td>
</tr>
</tbody>
</table>

**Authors (Partner)** Reinhold Herzog (Net Dynamics), Peter Zugmann (Net Dynamics), Joachim Quantz (Inubit)

**Responsible Author** Peter Zugmann  **Email** peter.zugmann@netdynamics-tech.com  
**Partner** Net Dynamics  **Phone** +43-1 503 98 26 16

**Abstract (for dissemination)**

**Keywords**

### Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-Jun-04</td>
<td>Draft 001</td>
<td>Peter Zugmann</td>
<td>First draft</td>
</tr>
<tr>
<td>08-Jun-04</td>
<td>Draft 002</td>
<td>Peter Zugmann</td>
<td>Changes after review of Michael Altenhofen and Sigurd Harand</td>
</tr>
<tr>
<td>21-Jun-04</td>
<td>Draft 003</td>
<td>Peter Zugmann</td>
<td>Changes after review of Christoph Mack</td>
</tr>
<tr>
<td>22-Jun-04</td>
<td>Draft 004</td>
<td>Alistair Duke</td>
<td>Language review changes</td>
</tr>
<tr>
<td>22-Jun-04</td>
<td>V1.00</td>
<td>Peter Zugmann</td>
<td>Final</td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>National University of Ireland Galway</td>
<td>NUIG</td>
<td>Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>National University of Ireland, Galway Galway Ireland Email: <a href="mailto:chris.bussler@deri.org">chris.bussler@deri.org</a> Tel: +353 91 512460</td>
</tr>
<tr>
<td>Fundacion De La Innovacion.Bankinter</td>
<td>Bankinter</td>
<td>Monica Martinez Montes Fundacion de la Innovation, Bankinter, Paseo Castellana, 29 28046 Madrid, Spain Email: <a href="mailto:mmtnez@bankinter.es">mmtnez@bankinter.es</a> Tel: 916234238</td>
</tr>
<tr>
<td>Berlecon Research GmbH</td>
<td>Berlecon</td>
<td>Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32 10117 Berlin, Germany Email: <a href="mailto:tw@berlecon.de">tw@berlecon.de</a> Tel: +49 30 2852960</td>
</tr>
<tr>
<td>British Telecommunications Plc.</td>
<td>BT</td>
<td>Dr John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham, Ipswich IP5 3RE, United Kingdom Email: <a href="mailto:john.nj.davies@bt.com">john.nj.davies@bt.com</a> Tel: +44 1473 609583</td>
</tr>
<tr>
<td>Swiss Federal Institute of Technology, Lausanne</td>
<td>EPFL</td>
<td>Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland Email: <a href="mailto:Karl.Aberer@epfl.ch">Karl.Aberer@epfl.ch</a> Tel: +41 21 693 4679</td>
</tr>
<tr>
<td>Essex County Council</td>
<td>Essex</td>
<td>Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. Email: <a href="mailto:maryr@essexcc.gov.uk">maryr@essexcc.gov.uk</a> Tel: +44 (0)1245 436524</td>
</tr>
</tbody>
</table>
| Forschungszentrum Informatik                 | FZI     | Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe, Germany Email: abecker@fzi.de Tel: +49 721 9654 0
## D4.2 Publishing process specification

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA | Christian de Sainte Marie  
9 Rue de Verdun, 94253  
Gentilly, France  
Email: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG | Torsten Schmale,  
inubit AG  
Lützowstraße 105-106  
D-10785 Berlin,  
Germany  
Email: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A. | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
Email: rbenjamins@isoco.com  
Tel. +34 913 349 797 |
| Net Dynamics Internet Technologies GmbH u. Co KG | Peter Smolle  
Net Dynamics Internet Technologies GmbH &. Co KG  
Prinz-Eugen-Strasse 68-70  
A-1040 Wien, Austria  
Email: peter.smolle@netdynamics-tech.com  
Tel.: +43 1 503982615 |
| The Open University | Dr. John Domingue  
Knowledge Media Institute,  
The Open University, Walton Hall,  
Milton Keynes, MK7 6AA,  
United Kingdom  
Email: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
Email: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma Al Ltd. | Atanas Kiryakov,  
Ontotext Lab, - Sirma AI EAD,  
Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse,  
Sofia 1784, Bulgaria  
Email: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Person</th>
<th>Address</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiscali Österreich GmbH</td>
<td>Dieter Haacker</td>
<td>Tiscali Österreich GmbH, Diefenbachgasse 35, A-1150 Vienna, Austria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="mailto:Dieter.Haacker@at.tiscali.com">Dieter.Haacker@at.tiscali.com</a></td>
<td>+43 1 899 33 160</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
<td>Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td>+972 2 6491111</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Carlo Wouters,</td>
<td>Vrije Universiteit Brussel, Pleinlaan 2, G-10 1050 Brussel, Belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Starlab- VUB</td>
<td></td>
<td><a href="mailto:carlo.wouters@vub.ac.be">carlo.wouters@vub.ac.be</a></td>
<td>+32 (0) 2 629 3719</td>
</tr>
</tbody>
</table>

**D4.2 Publishing process specification**
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY**........................................................................................................... 1  
**TABLE OF CONTENTS**............................................................................................................ VI  
**1 INTRODUCTION**...................................................................................................................... 1  
**2 PRINCIPLES AND APPROACH**.............................................................................................. 2  
  2.1 Central, de-central versus hybrid architecture......................................................................... 3  
  2.2 Semantically aware Registry versus semantically agnostic Registry........................................ 5  
  2.3 Recommended approach ......................................................................................................... 5  
**3 ARCHITECTURAL OVERVIEW**............................................................................................... 6  
  3.1 The Registry in the DIP Architecture..................................................................................... 6  
  3.2 Publishing Mechanism ............................................................................................................ 7  
  3.2.1 Access rights..................................................................................................................... 7  
  3.2.2 Scalability ......................................................................................................................... 7  
  3.2.3 Consistency....................................................................................................................... 7  
  3.3 Retrieval Mechanism ............................................................................................................. 8  
  3.3.1 Syntactical and structural retrieval – Registry.............................................................. 8  
  3.3.2 Semantic Retrieval - Discovery....................................................................................... 8  
  3.3.3 Access rights..................................................................................................................... 8  
  3.3.4 Scalability ......................................................................................................................... 8  
  3.3.5 Consistency ....................................................................................................................... 8  
**4 PUBLISHING AND RETRIEVAL INTERFACES**...................................................................... 8  
  4.1 Mapping DIP element properties into the Registry................................................................. 9  
  4.2 Publishing Interface ............................................................................................................... 9  
  4.3 Retrieval Interface ............................................................................................................... 12  
**5 PROPOSED IMPLEMENTATION OF DIP REGISTRY**............................................................. 15  
  5.1 Using DIP specific Registry or an existing one...................................................................... 16  
  5.2 UDDI as basis for a DIP Registry ............................................................................................ 16  
  5.2.1 UDDI Overview ................................................................................................................ 16  
  5.2.2 UDDI Data Model ............................................................................................................ 17  
  5.2.3 UDDI for DIP ................................................................................................................... 18  
  5.2.4 Coexistence and enhancement of existing Web Services.................................................. 18  
  5.3 Supported DIP elements ....................................................................................................... 19  
  5.4 Lifecycle of published information ......................................................................................... 19  
  5.5 UDDI Data Model Configurations ....................................................................................... 20  
  5.5.1 UDDI Categories Scheme ............................................................................................... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5.2 UDDI TModel-Keys</td>
<td>21</td>
</tr>
<tr>
<td>5.5.3 Mapping DIP element properties into the Registry</td>
<td>23</td>
</tr>
<tr>
<td>5.6 Interfacing UDDI</td>
<td>27</td>
</tr>
<tr>
<td>5.7 Publishing Interface</td>
<td>27</td>
</tr>
<tr>
<td>5.7.1 Publish DIP Web Service</td>
<td>27</td>
</tr>
<tr>
<td>5.7.2 Publish DIP Ontology</td>
<td>31</td>
</tr>
<tr>
<td>5.7.3 Publish DIP Goal</td>
<td>33</td>
</tr>
<tr>
<td>5.7.4 Publish DIP Mediator Service</td>
<td>34</td>
</tr>
<tr>
<td>5.8 Retrieval Interface</td>
<td>34</td>
</tr>
<tr>
<td>5.8.1 Retrieval Methods</td>
<td>34</td>
</tr>
<tr>
<td>5.8.2 Retrieval by non-DIP-aware Client</td>
<td>35</td>
</tr>
<tr>
<td>5.8.3 Retrieve DIP elements</td>
<td>35</td>
</tr>
<tr>
<td>5.8.4 Sample listings</td>
<td>36</td>
</tr>
<tr>
<td><strong>6 CONCLUSION</strong></td>
<td>39</td>
</tr>
<tr>
<td><strong>7 REFERENCES</strong></td>
<td>40</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

In the context of Web Services, the general idea of publishing a service is to collect all information needed or useful for its usage and to make it available to potential service users. As the fundamental idea of service-oriented architectures (SOA) is that services are entirely defined by their interfaces and the contracts they offer, these are the minimally required pieces of information which have to be provided when publishing a service.

Whereas the interface is usually specified in WSDL (Web Services Description Language) [3], before DIP, no formal language has been agreed upon yet to fully specify service contracts. In most cases, information concerning such contracts is provided as free text or by using proprietary sets of attributes. In DIP, the contract related information is semantically described in non functional properties [5]. Reusability is enhanced if all relevant information can be accessed via a central Repository or Registry (see below for a terminological distinction). In practice, many development projects still lack a rigorous approach towards using such a central Repository or Registry. As a consequence, relevant information about a service can often only be obtained by directly asking the developer who implemented it.

A well-defined publishing process can help to overcome this problem. The basic task of such a process is to define

1. the information to be collected when publishing a service
2. the format to be used to collect and store this information
3. the mechanism for actually storing the information in a Repository or a Registry

As DIP extends Web Services to Semantic Web Services, publishing in the context of DIP is not restricted to services alone but also comprises other so-called DIP elements, namely Ontologies and Goals. Mediator Services play an important role in DIP. From a Registry’s point of view, a Mediator Service is a special kind of Web Service. Unless stated otherwise, referring to Services also includes Mediator Services.

In this introduction, the presentation will focus on the publishing of services.

With respect to storing information, DIP distinguishes between the Repository, the physical database in which information is persistently stored, and the Registry, which is a component providing functionality for entering and retrieving information to and from the Repository. So creation of a DIP element is accomplished in a 2 phased approach: First the DIP element is created and deployed, which means that it is physically stored in a Repository. In a second step, the DIP element will be published to the Registry.

The decision on which information to include in the publishing of a service should be based on the various use case scenarios in which this information is to be used. The most important use cases are the following:

- Discovery of services
- (Semi-)automatic composition of services

The most obvious use of published information is discovery of services. This comprises two slightly distinct though related aspects. On the one hand, it involves the search for available services; on the other hand, it involves the detailed verification of whether a
found service exactly matches the requirements. Whereas the initial search might be based on rather generic criteria, the second step usually takes into account specific aspects of the non functional properties provided by a service.

A related but more complex use case is the (semi-)automatic composition of services. Here the focus is on the automatic evaluation of information available on services. Whereas “manual” discovery is feasible on the basis of free text information, standardized and structured formats are needed to make the automatic processing of published information possible.

The choice of information to be included when publishing a service ultimately depends on the specific application scenario. For example, a purchasing service will have a product category whereas a weather information service will not, but will need information about the location of interest. The publishing process thus has to provide a generic framework that can be adapted according to the specific needs of individual applications.

This document describes how Web Services, Ontologies and Goals (all together referred to as DIP elements in this document) can be published and retrieved in DIP using a Registry.

The document is structured as follows:

Chapter 2 explains the principles and the approach for designing the Registry;

Chapter 3 introduces the conceptual architecture of the Registry in DIP

Chapter 4 specifies the programming interfaces i.e. how to use a DIP Registry from an implementation neutral top-level view of publishing

Chapter 5 works out how a Registry could quickly be set up based on UDDI and how the concepts from chapter 4 can be mapped to such an implementation.

It is assumed, that the reader has a basic knowledge about the role of registries and repositories and, to understand the proposed implementation in detail, also needs to be familiar with the concepts of UDDI.

For a description of terms not explained in this document please refer to the DIP Glossary [12].

2 PRINCIPLES AND APPROACH

Publishing a DIP element basically consists of making information about it (publicly) available in order to allow its retrieval by potential consumers searching for it. Publishing is not concerned with the physical storage of DIP elements (which is the role of the Repository), only a reference to the stored elements is required and used. Of course the Registry must be able to access the location where the elements are stored to retrieve them.

Requirements for the Registry are to:

- Have a publish/retrieval procedure for Web Services, Goals and Ontologies (DIP elements)
- Reuse and adapt existing and well accepted standard methods and tools as opposed to 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 DIP Web Services

2.1 Central, de-central versus hybrid architecture

For DIP elements we have to differentiate between the DIP element itself (e.g. the executable service or the Goal), and its description (e.g. the WSDL interface). For both aspects of an element it has to be decided whether a centralized or a distributed approach for the Repository is preferable. In a centralized approach, all elements would be stored in a central Repository. Especially for the executable part of a service this is a difficult approach, since the service will hardly be able to execute from within the Registry and will in most cases run at the service provider’s location, where it has the required IT infrastructure at its command. Although the executable service could be stored in the Registry, it must either be a copy of the service from its execution location or it would have to be transferred to that location. As long as the executable code does not provide valuable information for users of the Registry there is no need to store it there.

Therefore, DIP elements themselves should be stored at the location where they are supposed to be executed.

A similar question arises for the description of a DIP element (e.g. the semantic information about a DIP Web Service). The description could be stored in a central description Repository or maintained in a decentralized manner, e.g. at the physical location of the DIP element itself.

![Diagram](image)

Figure 1: Centralized Registry/Repository approach

In the centralized approach, the description of each service is published to a central Registry. Because the discovery relies on the descriptions stored in the Registry, it is necessary to always keep the descriptions in the Registry up to date. To retrieve published elements, the Discovery can query a well known Registry. A Registry can even maintain information about other Registries (which would be published as a
service again). A central Registry (and there may be more than one Registry) has to be provided by a Registry Service Provider. Such a Registry Service Provider needs to be a trusted party and must offer a certain reliability since in this case the Registry is an important information-hub.

Figure 2: De-centralized approach

In the de-centralized approach, every service provider has its own Registry where it publishes information. The advantage of the de-centralized approach is that the Registry is not a single point of failure any more. The challenge for the discovery function is that it has to ensure that all providers are taken into account in a search.

Figure 3: Hybrid approach (“index” Registry)

In the hybrid approach, some abstract information is stored in a central Registry, while more specific service descriptions are stored locally at the providers. The abstract stored
in the central Registry should contain information detailed enough to allow for service identification and filtering in search and retrieval. In a first step, retrieval does not return the description itself, but only the abstract and a link to the detailed description. In a next step, the Retrieval mechanism can retrieve the detailed description. The Repository of the Registry can be organized either in a centralized or a distributed manner; a choice which is left to the Registry’s implementation. The hybrid approach combines the advantage of a central Registry (a known place where a Discovery element can search for descriptions of published items) and a de-central organized Repository, which makes maintenance and security of the actual items easier.

It should be noted that each approach offers certain advantages regarding the general trade-off between the following criteria:

- **Performance of the Registry**
  
  Response time of the Registry to a client heavily depends on how the Registry’s data is organized. Distributed data may take a long time to be collected to satisfy a query.

- **Network traffic**
  
  Centrally stored Registry data minimizes network traffic, but needs more sophisticated mechanisms to keep the information stored within the Registry up-to-date. Completely distributed data, especially when not cached and when every DIP element description detail has to be read from the network for each query, may generate unnecessarily high network traffic.

- **Complexity of consistency**
  
  When information is stored in a de-centralized manner, care must be taken to keep it consistent. If a Registry stores descriptions of DIP elements it must be assured that they reflect the current situation of the DIP elements themselves.

- **Ease of administration**
  
  A Registry with a central entry point for access is easier to maintain in respect to user access rights than a split Registry which must provide functions to replicate this information. Furthermore, the question of how easy it is to perform the upgrade of Registry requires consideration.

### 2.2 Semantically aware Registry versus semantically agnostic Registry

Another decision to be taken concerns the issue of whether to have a semantically aware Registry or not. If the Registry is semantically aware, the Retrieval must be able to deal with semantics and provide a semantic query interface. In a semantically agnostic Registry, some properties of the semantic descriptions have to be provided for use with a non-semantic retrieval interface.

### 2.3 Recommended approach

The recommended approach is to implement a DIP Registry following the hybrid model (a Registry containing abstract information, while detailed descriptions are stored de-centrally) and to clearly separate Repository, Registry and Discovery components in DIP. The Registry should be “ontology agnostic”, i.e. all semantic-related processing should be kept separate. Only as little information as necessary should be stored in the
Registry itself (the minimum that is required to perform a structural query), with references to detailed information stored elsewhere. In this approach the Registry can be viewed as an “index” to the storage. Although some recommendations are given in this document, it is up to the implementation to decide how much information about the stored data is copied and stored in the Registry itself.

3 ARCHITECTURAL OVERVIEW

3.1 The Registry in the DIP Architecture

For DIP, semantic retrieval is required. As identified in Chapter 2, the approach is to choose a hybrid approach for the Registry and clearly separate Registration, Repository and Discovery. The detailed recommendations are that:

- **DIP elements are physically stored de-centralized** (in most cases at the owner or provider of a DIP element). This process is also called deployment.
- Descriptions of the DIP elements including references to the physical locations of the elements are handed over to the Registry for publishing.
- The Registry stores information about the publisher and selected data about DIP elements. The Registry is “semantically agnostic” and contains links to the stored information, which itself contains the semantic information.
- The Registry provides a **non-semantic retrieval interface** where DIP elements can be retrieved using selected DIP element attributes.
- **Discovery** has “semantic knowledge” to discover DIP elements using the Registry to retrieve the location of the physical elements. Discovery provides a **semantic retrieval interface**.

Figure 4 shows how Registry, Repository and Discovery relate to each other.

![Proposed DIP Architecture for Registry, Discovery and Repository](image)

Figure 4: Proposed DIP Architecture for Registry, Discovery and Repository
Since Discovery, the semantic retrieval of DIP elements, is key in DIP, it is the driving point for the chosen architecture. A two-level approach is suggested:

1. Syntactical and structural retrieval (using standard Registry functions which are assumed to be provided by the Registry)

2. Semantic retrieval (using Discovery)

The main reasons for separating these two steps are:

- A clear separation of Registry and Discovery allows the development of several alternative Discovery mechanisms on top of the Registry; Discovery components may vary with respect to capabilities, functions and performance. Moreover, the Registry can be used by existing applications in a traditional, non-semantic way to retrieve DIP Web Services.

- For a Registry it is quite common and practicable to retrieve detailed information from distributed locations. A state-of-the-art reasoner in contrast would need all data locally in a special format to provide reasonable performance. This is true at least in the current stage since there is no scalable reasoner which is able to reason over data distributed across the net today.

### 3.2 Publishing Mechanism

DIP elements are published by providing information to the Registry that is needed for syntactical and structural retrieval. In addition, links to detailed information about DIP elements are provided in the publishing process. Some elements of the semantic description have to be mapped and converted to be used for structural retrieval. Later in this document we make a proposal, as to what properties of DIP elements should be mapped into the Registry (for details refer to chapter 4.1). In addition to the information that has to be published, the Registry must also maintain information about the Publishers.

Although Publishing and Retrieval are very closely related to each other, Retrieval is the more demanding, above all in terms of scalability.

#### 3.2.1 Access rights

Although Publishing may be open to everybody (depending on the policy of the Registry provider), registration of a Publisher should be mandatory in order to maintain the integrity of the Registry (this is quite easy for a central Registry).

#### 3.2.2 Scalability

Typically, users of a Registry will query items in the Registry much more often than they will publish new items or change existing ones. Scalability therefore is much more a matter for retrieval than for publishing (see chapter 3.3.4 below).

#### 3.2.3 Consistency

Care has to be taken regarding the consistency of published items if a published DIP element refers to another element, for example when checking whether a value in an Ontology is valid within a separately stored taxonomy. In particular, when the related elements are stored de-centrally in different locations a performance issue may arise.
3.3 Retrieval Mechanism

3.3.1 Syntactical and structural retrieval – Registry
DIP elements can be syntactically retrieved using the Registry’s standard API, taking into account the mapping that has been applied during publishing.

For an initial load from a Registry, the standard retrieval mechanisms of the Registry will be used. Search qualifiers are used together with the retrieval API to apply filters for technical and non-technical criteria. Only the stored information in the Registry can be used as filtering information - definitely not the semantic description of the Web Service.

3.3.2 Semantic Retrieval - Discovery
Discovery is responsible for the semantic retrieval of DIP elements.

An entry in the Registry, which is a structural description, points to a document containing detailed information including the semantics. It is up to the user to understand the content of the referred documents and is transparent from the point of view of the Registry. This content comprises Ontology descriptions, DIP Web Service descriptions and Goal descriptions.

3.3.3 Access rights
Although a Registry provider may decide differently, it is recommended to allow any user to query the Registry.

3.3.4 Scalability
The proposal allows centralized and distributed Registries, and specialized Discovery services. Scalability can be achieved through partitioning.

For the design of a scalable production implementation, the number of published elements, potential users and programs as well as the expected frequency of usage has to be taken into account. In addition, the change frequency of the Registry content is important, especially if Goals with short life cycles are to be registered.

To improve scalability, the Registry may cache the DIP element descriptions for faster access (but it then has to care that the cache is up-to-date). A Discovery Service may share the same cache and as a result the data format could then be optimized for the Discovery Service.

For a Discovery mechanism it will definitely be better to retrieve all DIP element descriptions in bulk from the Registry as opposed to issuing a high number of single requests.

3.3.5 Consistency
Another major issue is the consistency and contemporary nature of the Discovery service. It is up to the implementer of a discovery service to choose the strategy which best fits the balance between costs of requesting updates from the Registry and dealing with potentially outdated or missing information.

4 Publishing and retrieval interfaces
This chapter specifies high level publishing and retrieval interfaces which are independent from the implementation of a Registry.
Since we will propose to implement a DIP Registry on top of an existing tool, only functions that are specific to publishing and retrieving DIP elements are specified in this document. Other required functions, e.g. maintenance, are not covered here and can be found in the documentation of the underlying Registry tool.

The interface description is shown in the WSDL format as a sample. Some specifications, such as the binding, have to be worked out in detail for a specific implementation.

4.1 Mapping DIP element properties into the Registry

When publishing a DIP element, it is proposed that the following properties should be stored within the Registry (the element properties are taken from the WSMO non functional properties [6]):

<table>
<thead>
<tr>
<th>Mapped DIP element properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
</tr>
<tr>
<td>identifier</td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>publisher</td>
</tr>
<tr>
<td>creator</td>
</tr>
<tr>
<td>coverage</td>
</tr>
<tr>
<td>version</td>
</tr>
<tr>
<td>language</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>format</td>
</tr>
<tr>
<td>source</td>
</tr>
<tr>
<td>relation</td>
</tr>
<tr>
<td>grounding (URN, Service and Mediator Service only)</td>
</tr>
<tr>
<td>subject</td>
</tr>
</tbody>
</table>

4.2 Publishing Interface

The service for publishing provides operations to publish a DIP Web Service, a Mediator Service, an Ontology and a Goal.

Two parameters are common for each operation: The Business Entity that publishes the DIP element and a link to the file which contains the detailed description of the DIP element (in the WSDL example below called WNFP). In addition to that, a DIP Web Service and a Mediator Service take the functional service description (WSDL) as input. The publishing program is expected to read the description files and store the required properties in the Registry’s database.
Listing 1: Publishing Interface Web Service Description

```xml
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/
    xmlns:tns="urn:dip-org:publish_v1"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:dip-org:publish_v1"
    name="DIP_Publish_API_V1">
    <documentation>
        WSDL Service Interface for DIP element publishing API V1.0
    </documentation>

    <types>
        <xsd:schema targetNamespace="urn:dip-org:publish_v1"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:Type name="businessEntityType" type="xsd:string" minOccurs="1"/>
            <xsd:Type name="serviceKeyType" type="xsd:token"/>
            <xsd:Type name="ontologyKeyType" type="xsd:token"/>
            <xsd:Type name="goalKeyType" type="xsd:token"/>
            <xsd:Type name="authTokenType" type="xsd:token"/>
            <xsd:Type name="userIdType" type="xsd:token"/>
            <xsd:Type name="credentialType" type="xsd:token"/>
            <xsd:complexType name="dipElementPropertiesAbstractType">
                <xsd:sequence>
                    <xsd:element name="title" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="identifier" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="description" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="publisher" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="creator" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="coverage" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="language" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="date" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="format" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="source" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="relation" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="subject" minOccurs="0" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="result">
                <xsd:sequence>
                    <xsd:element name="code" minOccurs="1" type="xsd:string"/>
                    <xsd:element name="description" minOccurs="0" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="resultListType">
                <xsd:sequence>
                    <xsd:element name="result" minOccurs="1" maxOccurs="unbounded" type="tns:result"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </types>

    <message name="publishServiceDescription">
        <part name="authToken" type="tns:authTokenType"/>
        <part name="serviceKey" type="tns:serviceKeyType"/>
        <part name="businessEntity" type="tns:businessEntityType"/>
        <part name="dipElementPropertiesAbstract" type="tns:dipElementPropertiesAbstractType"/>
    </message>
</definitions>
```
<part name="wnfp" type="xsd:anyURI"/>
<part name="wsdl" element="xsd:anyURI"/>
</message>

<message name="publishOntologyDescription">
<part name="authToken" type="tns:authTokenType"/>
<part name="ontologyKey" type="tns:ontologyKeyType"/>
<part name="businessEntity" type="tns:businessEntityType"/>
<part name="dipElementPropertiesAbstract" type="tns:dipElementPropertiesAbstractType"/>
<part name="wnfp" type="xsd:anyURI"/>
</message>

<message name="publishGoalDescription">
<part name="authToken" type="tns:authTokenType"/>
<part name="goalKey" type="tns:goalKeyType"/>
<part name="businessEntity" type="tns:businessEntityType"/>
<part name="dipElementPropertiesAbstract" type="tns:dipElementPropertiesAbstractType"/>
<part name="wnfp" type="xsd:anyURI"/>
</message>

<message name="deleteServiceKey">
<part name="authToken" type="tns:authToken" />  
<part name="serviceKey" type="tns:serviceKeyType" />  
</message>

<message name="deleteOntologyKey">
<part name="authToken" type="tns:authToken" />  
<part name="ontologyKey" type="tns:ontologyKeyType" />  
</message>

<message name="deleteGoalKey">
<part name="authToken" type="tns:authToken" />  
<part name="goalKey" type="tns:goalKeyType" />  
</message>

<message name="authTokenMsg">
<part name="authToken" type="tns:authToken" />  
</message>

<message name="logon">
<part name="userId" type="tns:userIdType" />  
<part name="credential" type="tns:credentialType" />  
</message>

<portType name="Publish">
<documentation>
This portType defines all of the dip publish operations.
</documentation>

<operation name="dipPublishService">
<input message="tns:publishServiceDescription"/>
<output message="tns:publishServiceKey"/>
<fault name="error" message="tns:resultList"/>
</operation>

<operation name="dipPublishMediatorService">
<input message="tns:publishServiceDescription"/>  
<output message="tns:publishServiceKey"/>  
<fault name="error" message="tns:resultList"/>  
</operation>

<operation name="dipPublishOntology">
<input message="tns:publishOntologyDescription"/>  
<output message="tns:publishOntologyKey"/>  
</operation>
4.3 Retrieval Interface

The service for retrieval provides operations to retrieve either DIP Web Services, Mediator Services, Ontologies or Goals according to certain search criteria.

The inputs for retrieval are key value pairs. The keys are the DIP element properties and the publisher information listed in the Publishing Interface in chapter 4.2.

For a query, a find qualifier AND, OR or LIKE can be specified. Also wildcard symbols can be used (limited to the title property). If no find qualifier is specified, AND is used as default. To avoid inventing a completely new query language, the interpretation and behaviour of the find qualifiers are supposed to work as specified in UDDI V2 [1].
The result is a list of DIP elements with the DIP element properties stored in the Registry and links to the documents containing the detailed information.

**Listing 2: Retrieval Interface Web Service Description**

```xml
<definitions
 xmlns="http://schemas.xmlsoap.org/wsdl/
 xmlns:tns="urn:dip-org:retrieve_v1"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
 targetNamespace="urn:dip-org:retrieve_v1"
 name="DIP_Retrieve_API_V1">
 <documentation>
 WSDL Service Interface for DIP element retrieval API V1.0
 </documentation>
 <types>
 <xsd:schema targetNamespace="urn:dip-org:retrieve_v1"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:simpleType name="dipFindOrLikeQualifierType">
 <xsd:annotation>
 <xsd:documentation>default is AND</xsd:documentation>
 </xsd:annotation>
 <xsd:restriction base="xsd:string">
 <xsd:enumeration value="AND">
 <xsd:annotation>
 <xsd:documentation>all element properties specified must match,
 unlikely to work for multiple subjects</xsd:documentation>
 </xsd:annotation>
 </xsd:restriction>
 <xsd:enumeration value="OR">
 <xsd:annotation>
 <xsd:documentation>any element property match, perhaps not
 wanted for {date=20040419},{subject=travel},{subject=tickets}
 </xsd:documentation>
 </xsd:annotation>
 </xsd:restriction>
 <xsd:enumeration value="LIKE">
 <xsd:annotation>
 <xsd:documentation>all element properties match within the
 tModel/{date=20040419},{subject=travel},{subject=tickets}-
 match DATE and either TICKETS or TRAVEL
 </xsd:documentation>
 </xsd:annotation>
 </xsd:restriction>
 </xsd:simpleType>
 <xsd:complexType name="dipElementPropertiesAbstractType">
 <xsd:sequence>
 <xsd:element name="title" minOccurs="0" type="xsd:string"/>
 <xsd:element name="identifier" minOccurs="0" type="xsd:string"/>
 <xsd:element name="description" minOccurs="0" type="xsd:string"/>
 <xsd:element name="publisher" minOccurs="0" type="xsd:string"/>
 <xsd:element name="creator" minOccurs="0" type="xsd:string"/>
 <xsd:element name="coverage" minOccurs="0" type="xsd:string"/>
 <xsd:element name="language" minOccurs="0" type="xsd:string"/>
 <xsd:element name="date" minOccurs="0" type="xsd:string"/>
 <xsd:element name="format" minOccurs="0" type="xsd:string"/>
 <xsd:element name="source" minOccurs="0" type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>
```
5 Proposed Implementation of DIP Registry

Although the most flexible and customized approach would be to build a DIP Registry from scratch, this document makes a proposal to implement a DIP Registry based on available tools. A lot of specification and implementation work has gone into
development of these tools, also covering aspects of access rights, maintenance, versioning and other necessary topics which must be provided by a DIP Registry, but are not specific DIP requirements.

In chapter 4, implementation independent interfaces for publishing and retrieval have been specified. This chapter elaborates, how the concepts from chapter 4 can be mapped to a specific Registry implementation. It will be shown, how publishing and retrieval can be carried out using the interfaces of a UDDI Registry [1]. For a real implementation, the high-level interface methods defined in chapter 4 should be implemented by the appropriate sequence of the listed UDDI function calls below.

5.1 Using DIP specific Registry or an existing one

Currently there are no Registry implementations available that support semantics. If an already implemented Registry should be used, support for semantic descriptions has to be added. As elaborated in chapter 2.3, the Registry itself should remain semantics agnostic, so the requirement for a Registry is to be able to publish and retrieve items with semantic content, but not to be able to “understand” semantics, particularly when queried.

A Registry, specially customized according to DIP needs would of course be a best fit. On the other hand, a Registry needs a lot of basic functions like storage, maintenance and access rights which require a large effort to design, but would not add value to DIP. The proposal is to base the DIP Registry on an existing implementation.

In DIP deliverable D4.1 [11], existing state-of-the-art Registry implementations are listed: UDDI [1], ebXML [8], and WSIL [9]. Although very rich in functions, the ebXML Registry is very much related to concepts and components of the ebXML specification. WSIL is very open at the conceptual level and also follows a more decentralized approach, but its very simple mechanism – a service is published by putting a description file in a directory that follows certain naming conventions – offers very limited possibilities for extensions, above all in the area of security.

UDDI seems to be the best trade off regarding architecture, openness and flexibility. Also, UDDI is quite well accepted and free implementations are available. Last but not least, UDDI to a great extent fulfills the requirements listed in DIP deliverable D4.1, and, as such, has been chosen as the basis for the DIP registry.

5.2 UDDI as basis for a DIP Registry

5.2.1 UDDI Overview

Together with SOAP [10] and WSDL, UDDI (Universal Description, Discovery and Integration) is one of the core technologies that enable Web Services. UDDI provides mechanisms to publish and retrieve Web Services and contains categorized information about businesses and the services that they offer. The services are associated with technical specifications of the Web Service and these technical specifications are usually defined in WSDL. To find out how a Web Service has to be used, the UDDI Registry will be queried to find the WSDL descriptions.

The UDDI project [13] operates as a global public Registry. The service is free and all information stored in the Registry is public. In addition to that, organizations can set-up a private Registry to support, for example, the requirements of an enterprise.
5.2.2 UDDI Data Model

The core information model of UDDI defines four different types of information:

**Business Entity**

Holds information about the party who publishes information about a service. This is similar to information in White Pages.

**Business Service**

This is information about a particular family of technical services. This is similar to information in Yellow Pages. Business services are linked to Business Entities.

**Binding Template**

A Binding Template holds information about a service entry point and construction specifications. This is similar to information in Green Pages. A Business Service contains binding templates.

**tModels**

These describe specifications for services or taxonomies. These form the basis for technical fingerprints. Binding template data contains references to tModels. These references designate the interface specifications for a service.

Category Bags and Identifier Bags, providing key-value pairs can be assigned to UDDI’s main information types to be able to specify additional search criteria:

**Category Bag**

Categories, provided in Category Bags allow the categorization of businessEntities, businessServices, and tModels to occur according to any of several available taxonomy-based classification schemes. Some schemes are already provided in UDDI implementations, for example industry codes (via NAICS [1]) or product and service classifications (via UNSPSC [1]). UDDI V2 allows custom categorization schemes.

**Identifier Bag**

Identifiers, provided in Identifier Bags, allow businessEntities and tModels to include information about common forms of identification such as numbers, tax identifiers, etc.
5.2.3 UDDI for DIP

DIP elements will be published to UDDI using standard functions. It is not necessary to extend the data model of UDDI (which is possible in UDDI V3 [2]). In fact necessary DIP element properties are mapped into existing data slots in UDDI. This keeps the Registry “semantically agnostic” and allows the use of UDDI versions prior to V3.

Later in this document it will be described how UDDI should be used in DIP and how properties of DIP elements can be mapped into the UDDI data model so that UDDI can be used for retrieval using its standard find functions.

In its specification, UDDI V3 provides mechanisms to extend UDDI’s data model as well as its API. Since no UDDI V3 implementation is currently readily available, it was decided to base the DIP approach on concepts and functions available in UDDI V2.²

5.2.4 Coexistence and enhancement of existing Web Services

A UDDI Registry can be used for both existing Web Services and DIP elements at the same time. DIP elements will be published and retrieved using special tModels, whereas existing Web Services are open to use their own ones.

However, the interfaces to UDDI differ. The native UDDI interface functions are used by existing UDDI clients, whereas DIP aware clients publish and retrieve using the DIP Publishing and Retrieval interface as defined in chapter 4. The latter interface itself is a UDDI client and uses its standard functions.

Figure 6: DIP aware clients and standard UDDI clients share the Registry

If appropriate, an already existing Web Service can be extended with DIP semantic descriptions. In this case, the entries in the UDDI Registry have to be migrated. On the other hand, DIP Web Services can be found by any client that uses the standard UDDI retrieval interface. The examples below show how search criteria can be provided

---

² DIP will continue close monitoring of standardization processes with respect to UDDI. Once implementations of UDDI v3 are readily available, a migration of the DIP Registry will be considered.
during publishing so existing, non DIP, retrieval clients are able to find DIP Services via the use of keywords.

5.3 Supported DIP elements

The DIP Registry can be used to publish and retrieve the following DIP elements:

Web Services

Both, the DIP element properties (Non-Functional Properties etc.) and the Grounding (WSDL) can be published to the Registry. Because of the separation of the Grounding from the element properties it is possible to extend existing, already published services with DIP properties and to stay compatible with standard UDDI functions.

Mediator Service

Mediator Services will be treated as normal Web Services. However, they will be represented by a separate tModel implementation in UDDI which makes retrieval easier.

Ontologies

Ontologies can also be published and retrieved. In contrast to services, the description of an Ontology is the Ontology itself.

Goals

Goals have a much more “personal” flavour than other DIP elements. A Goal in most cases expresses the “wish” of a user and therefore may contain sensitive or confidential data. Nevertheless some of these Goals may be reusable and provided as templates and therefore should be published using the Registry. It has to be taken into consideration that these Goals should not be subject to very frequent changes (especially in UDDI V3, with it’s replication facilities). Non-template Goals, which are in most cases not reusable, frequently updated, and do not exist for very long, should not be published to the Registry. Storage of non-template Goals is not within the scope of this document.

To avoid registering non-suitable Goals, publishing rights should only be granted to trusted parties (e.g. template providers).

As with Ontologies, the description of a Goal is currently identical with the Goal itself.

It has to be noted, that for Goal retrieval, all rules, limitations and effects regarding visibility and security apply as specified in UDDI.

5.4 Lifecycle of published information

Since information entries in the Registry may sometimes reference each other, special care has to be taken to keep them consistent. It has, for example, to be avoided that services refer to Ontologies or Mediators that do not exist any more.

Due to the structure of UDDI, special tModels are created that do not refer to a defined DIP Web Service, Ontology or Goal. To simplify deletion of such tModels for a specific Business Entity it is proposed to add an Identifier Bag entry with tModel dipBusinessEntity, name businessEntity and value dipAdmin or similar.

An Ontology is registered as a tModel and can therefore survive the removal of the Business that published the Ontology – the Ontology would then become owner-less
and difficult to maintain. As with all UDDI interactions, a publish action may require an authInfo which, depending on the UDDI policy, requires a registered Business relationship with the UDDI provider to edit the Ontology.

Published services and Mediators become unavailable after the Business Entity is deleted. Published Goals still remain available after the Business Entity is deleted. This can be avoided in a future API wrapper.

5.5 UDDI Data Model Configurations

The UDDI standard data model without extensions is used.

5.5.1 UDDI Categories Scheme

UDDI V2 does not allow searching for a service by its identifier which would be convenient to retrieve all DIP elements of a certain type. To overcome this restriction it is proposed to add a categorization scheme of DIP elements. As a starting point, the following categories should be added.

dipElement
This is the root category of the categories scheme for the DIP categories listed below to support the hierarchical structure. A DIP element cannot be registered using that category, but the category can be used to retrieve all DIP elements.

dipCategory
Used to retrieve all tModels that refer to a DIP element.

dipBusinessEntity
This category allows searching for all Business Entities that own DIP elements.

dipWebService
Categorizes a UDDI entry as DIP Web Service.

dipOntology
Categorizes a UDDI entry as Ontology

dipGoal
Categorizes a UDDI entry as Goal

dipMediatorService
Categorizes a UDDI entry as DIP Mediator Service

Usually, the schema has to be imported into the UDDI. The format depends on the UDDI implementation, below is an example in the Microsoft UDDI format:

Listing 3: UDDI Categorization schema

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:uddi="urn:uddi-org:api_v2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```
xmlns="urn:uddi-microsoft-com:api_v2_extensions">
</resources>

5.5.2 UDDI TModel-Keys

For DIP elements new, required tModels are introduced:

- dipWebService
- dipOntology
- dipGoal
- dipMediatorService
- dipBusinessEntity

This tModel is used to allow searching for all tModels that relate to a specific Business Entity; if this tModel is used in an Identifier Bag, the value is the same as the name of businessEntity.

- dipSubject

For each DIP element property, that may have multiple values (like Subject that will be mapped into a UDDI Category Bag (see chapter 5.5.3 below)), a separate tModel has to be created. This is only the case when a search using the find qualifier “LIKE” is required to allow a query such as “date must be 2004/04/20 and subject is either travel or tickets or both”. Without such a tModel travel or tickets would be found for all dates. Currently only Subject is such a property.
For each tModel, a category *dipCategory* is added to be able to retrieve all tModels that refer to a DIP element (for example for deletion of all DIP elements).

**Listing 4: Create tModels sample**

```xml
<save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
  <authInfo>OBNQxY5jV...drPZf</authInfo>
  <tModel tModelKey=""><name>dipBusinessEntity</name>
    <description>Owner for a tModel, e.g. goal, Ontology</description>
    <overviewDoc>
      <overviewURL>http://.../dipBusiness.html</overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference tModelKey="uddi-org:types" keyName="types"
        keyValue="relationship"/>
      <keyedReference tModelKey="uddi-org:types" keyName="types"
        keyValue="namespace"/>
      <keyedReference tModelKey="uddi-org:relationships"
        keyName="DIP Category Classification Scheme" keyValue="1.2"/>
      <keyedReference tModelKey="DIP Category Classification Scheme"
        keyName="urn:dip:category" keyValue="1.1"/>
    </categoryBag>
  </tModel>
  <tModel tModelKey=""><name>dipService</name>
    <description>DIP Service tModel</description>
    <overviewDoc>
      <overviewURL>http://.../dipService.html</overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference tModelKey="uddi-org:types" keyName="types"
        keyValue="specification"/>
      <keyedReference tModelKey="uddi-org:types" keyName="types"
        keyValue="signatureComponent"/>
      <keyedReference tModelKey="uddi-org:relationships"
        keyName="DIP Category Classification Scheme" keyValue="1.3"/>
      <keyedReference tModelKey="DIP Category Classification Scheme"
        keyName="urn:dip:category" keyValue="1.1"/>
    </categoryBag>
  </tModel>
  <tModel tModelKey=""><name>dipOntology</name>
    <description>DIP Ontology tModel</description>
    <overviewDoc>
      <overviewURL>http://.../dipOntology.html</overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference tModelKey="uddi-org:types" keyName="types"
        keyValue="specification"/>
      <keyedReference tModelKey="uddi-org:relationships"
        keyName="DIP Category Classification Scheme" keyValue="1.4"/>
      <keyedReference tModelKey="DIP Category Classification Scheme"
        keyName="urn:dip:category" keyValue="1.1"/>
    </categoryBag>
  </tModel>
  <tModel tModelKey=""><name>dipGoal</name>
    <description>DIP Goal tModel</description>
    <overviewDoc>
      <overviewURL>http://.../dipGoal.html</overviewURL>
    </overviewDoc>
    <categoryBag>
```
5.5.3 Mapping DIP element properties into the Registry

A meaningful set of DIP element properties have to be mapped into UDDI to meaningfully retrieve DIP elements in UDDI using UDDI’s limited find capability. This document makes a proposal, that DIP element properties should be mapped in a first attempt. A subset of the WSMO Core Non Functional Properties [6] has been selected. Mapping of additional attributes of the core NFPs or Non Functional Properties specific for Web Services (like performance or robustness) may be added.

Due to UDDI’s simple data model, the semantic context of the properties will get lost during the mapping.

Mapping DIP element properties into UDDI

Some DIP element properties are mapped directly into an existing data slot in UDDI, for example the property title of the Web Service into the name of the businessService.
The remaining properties are either mapped into Category Bags or Identifier Bags. When possible, Categories are preferred (because they can be internally checked). If a DIP element property cannot be categorized at all (like description) it goes into the Identifier Bag.

The following table shows which of the mapped DIP elements go into the Category Bag and which go into the Identifier Bag.

<table>
<thead>
<tr>
<th>DIP Element Property</th>
<th>Category</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>publisher</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>creator</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>version</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>coverage</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>language</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>date</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>format</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>source</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>relation</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>subject</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

The publisher of a DIP element is a special case. UDDI’s data model has a businessEntity, which is a good place to put the publisher’s name. For corporate standard reasons the publisher may be different from the Business Entity. Therefore we recommend that both the publisher and the Business Entity (which in most cases have the same values) should be maintained. The name of the Business Entity will be provided during publishing.

Since a businessService does not have Identifier Bags, a tModel for each service will be created and the Identifier Bag of the tModel will be used as a target.

In the figures below, the mapping of the properties into the different target locations is shown. The mappings of only some properties are shown in the examples:
Figure 7: Mapping a DIP Web Service to UDDI

* title is mapped to Name of all WSDL bindings
* description is mapped to all WSDL descriptions
Figure 8: Mapping a DIP Goal to UDDI

Figure 9: Mapping a DIP Ontology to UDDI

**Property transformation**

To map DIP element properties into a UDDI Bag, a transformation of the property value may be necessary. For DIP element properties that may have multiple values (like *subject* or *creator*), a Category Bag entry with the same key has to be created for each value. For example the property *subject* as described in the WSMO Primer [7]

```plaintext
ONFP:/> title ->> (Plane Itinerary Ontology)
creator ->> {Sinuhe Arroyo}
subject ->> {travel, leisure, plane}
```
would be split into 3 UDDI Category Bags, each containing for example the key `urn:wsmo:nfps:subject` and one of the terms `travel`, `leisure` and `plane`.

**Language property**

Descriptions in UDDI may have a language which must be ISO-3266 compliant. If the language can be retrieved from the DIP element properties in the appropriate format (not English), the mapping can be done automatically.

**Mapping the Groundings**

As long as there is no specification within DIP for grounding, a WSDL must always be created on top and a reference must be established from UDDI to the WSDL.

**Additional mappings**

In addition to the proposed mappings of selected DIP element properties, other properties may be added. For example, some of the functional properties would qualify for that (like the grounding type).

5.6 Interfacing UDDI

UDDI provides a publishing and retrieval API which has its peculiarities. For DIP, these interface are wrapped by the interface functions described in Chapter 4 and therefore are transparent to the user of the DIP publishing and retrieval functions. Nevertheless, the native UDDI API will allow non-DIP-aware programs to retrieve DIP elements.

Maintenance of the Registry has to be performed using UDDI’s set of functions.

5.7 Publishing Interface

In the procedures described below it is assumed that a Categorization scheme as described in chapter 5.5.1 has been created in or imported to the UDDI.

5.7.1 Publish DIP Web Service

Publishing a DIP Web Service in the Registry includes the following steps:

1) **Define used Categorization Scheme**

<table>
<thead>
<tr>
<th>Category Bag Entry</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization Scheme</td>
<td>&quot;DIP Category Classification Scheme&quot;</td>
</tr>
<tr>
<td>Key Name</td>
<td>&quot;dipWebService&quot;</td>
</tr>
<tr>
<td>Key Value</td>
<td>&quot;1.3&quot;</td>
</tr>
</tbody>
</table>

2) **Define the owner of the Service**

   The owner of a service, expected to be the publisher’s name, will be stored in the Identifier Bag to be able to retrieve a service by owner.
Table 4: Required Identifier Bag Entry

<table>
<thead>
<tr>
<th>Identifier Bag Entry</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Scheme</td>
<td>&quot;dipBusinessEntity&quot;</td>
</tr>
<tr>
<td>Key Name</td>
<td>&quot;businessEntity&quot;</td>
</tr>
<tr>
<td>Key Value</td>
<td>Name of Business Entity</td>
</tr>
</tbody>
</table>

For each DIP element property mapped into the UDDI, an Identifier Bag has to be added. Here is an example for the element property “language”:

Table 5: DIP element property Identifier Bag Entry

<table>
<thead>
<tr>
<th>Identifier Bag Entry</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization Scheme</td>
<td>&quot;uddi-org:general_keywords&quot;</td>
</tr>
<tr>
<td>Key Name</td>
<td>&quot;urn:dip:nfps:language&quot;</td>
</tr>
<tr>
<td>Key Value</td>
<td>&quot;English&quot;</td>
</tr>
</tbody>
</table>

Additional Identifier Bag entries may be added to allow retrieval in non-DIP UDDI usage.

3) Create Binding Template for the Business Service

Table 6: Binding Template

<table>
<thead>
<tr>
<th>Binding Template</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessPoint</td>
<td>URL of the DIP document</td>
</tr>
<tr>
<td>Description</td>
<td>Description from DIP-core element properties</td>
</tr>
<tr>
<td>Type</td>
<td>&quot;other&quot;</td>
</tr>
</tbody>
</table>

4) Create tModel

For each service, a separate tModel must be created. Among other things, this will allow searching by Identifier Bag.

Table 7: UDDI tModel for DIP Web Service

<table>
<thead>
<tr>
<th>tModel Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>title from DIP element properties</td>
</tr>
<tr>
<td>Description</td>
<td>description from DIP element properties</td>
</tr>
<tr>
<td>Categories</td>
<td>Category Bag as specified in step 1</td>
</tr>
<tr>
<td>Identifiers</td>
<td>Identifier Bag as specified in step 2</td>
</tr>
<tr>
<td>Overview Document</td>
<td>URL of the DIP Web Service element properties</td>
</tr>
</tbody>
</table>

5) Create UDDI Business Service

Table 8: UDDI Business Service

<table>
<thead>
<tr>
<th>Business Service Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Service name</td>
<td>title from DIP element properties</td>
</tr>
</tbody>
</table>
**6) Add HTTP WSDL Binding**

Add HTTP WSDL Binding as normal. This is especially important as non-DIP-aware clients will search for, resolve and invoke the service through this binding. This binding must therefore be self-sufficient and not require any DIP element properties.

Adding a binding is not described here in detail, but the following data to create the binding should be used:

- URL of DIP element properties
- Location URL of WSDL location
- Category Bag as specified in step 1
- Identifier Bag as specified in step 2

**7) Add Instance for the DIP Binding**

<table>
<thead>
<tr>
<th>Table 9: Instance for DIP Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DIP Binding</strong></td>
</tr>
<tr>
<td>Interface tModel:</td>
</tr>
<tr>
<td>Instance Details:</td>
</tr>
<tr>
<td>Overview Document:</td>
</tr>
</tbody>
</table>

Re-publishing a DIP Web Service is similar to publishing a new service, but requires the Service key, and may require the preserved Bags.

**8) Sample listings**

**Listing 5: Service registration sample**

```xml
<save_service generic="2.0" xmlns="urn:uddi-org:api_v2">
  <authInfo>OBNQxY5jV...Ow9P2VIo=/authInfo>
  <businessService serviceKey="" businessKey="7ba1cdc2-b97a-4ed2-9443-84271fd091fb">
    <name xml:lang="en">Sample Train Booking Service</name>
    <description>Train Itineraries for Online Ticket Booking</description>
    <categoryBag>
      <keyedReference tModelKey="DIP Category Classification Scheme"
                      keyName="urn:dip:service" keyValue="1.3"/>
      <keyedReference tModelKey="dipBusinessEntity"
                      keyName="urn:dip:nfps:publisher" keyValue="VTA"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:version" keyValue="1.0"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:coverage" keyValue="Europe"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:language" keyValue="English"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:date" keyValue="20040430"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:format" keyValue="text"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="travel"/>
```
Listing 6: Service registration sample, tModel creation

```xml
<save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
  <authInfo>OBNQxY5jV6viK...aPj3Ry/xvIZxk8=/authInfo>
  <tModel tModelKey="">
    <name>Sample DIP Service tModel</name>
    <overviewDoc>
      <overviewURL>http://.../doc.xml</overviewURL>
    </overviewDoc>
    <identifierBag>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:identifier"
        keyValue="http://.../dt.flr"/>
      <keyedReference tModelKey="dipBusinessEntity"
        keyName="urn:dip:nfps:businessEntity"
        keyValue="DERI"/>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:creator"
        keyValue="Max Mustermann"/>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:source"
        keyValue="http://.../resources/dt.flr"/>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:relation"
        keyValue="http://www.isi.edu/~pan/damltime/time-entry.owl"/>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:relation"
        keyValue="http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock"/>
      <keyedReference tModelKey="dipService"
        keyName="urn:dip:nfps:relation"
        keyValue="http://www.w3.org/TR/xmlschema-2"/>
    </identifierBag>
    <categoryBag>
      <keyedReference tModelKey="DIP Category Classification Scheme"
        keyName="urn:dip:service"
        keyValue="1.3"/>
    </categoryBag>
  </tModel>
</save_tModel>
```

Listing 7: Service registration sample, bindingTemplate creation

```xml
<save_binding generic="2.0" xmlns="urn:uddi-org:api_v2">
  <authInfo>OBNQxY5jV6viK...aPj3Ry/xvIZxk8=/authInfo>
  <bindingTemplate bindingKey="" serviceKey="484df54f-ff67-4b7c-a4e1-
    eb4c923b631">
    <accessPoint URLType="other">//boreasnet:8080/temp</accessPoint>
    <tModelInstanceDetails>
      <tModelInstanceInfo tModelKey="uuid:bf0a0d40-3f04-4f1c-999b-
        5f5cd96e89e">
        <InstanceDetails>
          <overviewDoc>
```
5.7.2 Publish DIP Ontology

Publishing a DIP Ontology in the Registry includes the following steps:

1) Define used Categorization Scheme

Table 10: Required Category Bag Entry

<table>
<thead>
<tr>
<th>Category Bag Entry</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization Scheme</td>
<td>&quot;DIP Category Classification Scheme&quot;</td>
</tr>
<tr>
<td>Key Name</td>
<td>&quot;dipOntology&quot;</td>
</tr>
<tr>
<td>Key Value</td>
<td>&quot;1.4&quot;</td>
</tr>
</tbody>
</table>

Additional Category Bag entries may be added to allow retrieval in non-DIP UDDI usage.

2) Define the owner of the Ontology

The owner of an ontology, expected to be the publisher’s name, can be stored in the Identifier Bag to enable retrieval of an Ontology by its owner.

Table 11: Required Identifier Bag Entry

<table>
<thead>
<tr>
<th>Identifier Bag Entry</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Scheme</td>
<td>&quot;dipBusinessEntity&quot;</td>
</tr>
<tr>
<td>Key Name</td>
<td>&quot;businessEntity&quot;</td>
</tr>
<tr>
<td>Key Value</td>
<td>Name of the Business Entity</td>
</tr>
</tbody>
</table>

Additional Identifier Bag entries may be added to allow retrieval in non-DIP UDDI usage.

3) Create tModel

For each Ontology, a separate tModel must be created

Table 12: UDDI tModel for DIP Ontology

<table>
<thead>
<tr>
<th>tModel Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td><em>title</em> from DIP element properties</td>
</tr>
<tr>
<td>Description:</td>
<td><em>description</em> from DIP element properties</td>
</tr>
<tr>
<td>Categories:</td>
<td>Category Bag as specified in step 1</td>
</tr>
<tr>
<td>Identifiers:</td>
<td>Identifier Bag as specified in step 2</td>
</tr>
<tr>
<td>Overview Document:</td>
<td>URL of the DIP Ontology</td>
</tr>
</tbody>
</table>

Re-publishing a DIP Ontology is similar to publishing a new Ontology, but requires the Ontology key, and may require the preserved Bags.
4) Sample listing

Listing 8: Ontology registration sample

```xml
<save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
  <authInfo>OBWO...zm+E1xRl03WOIc</authInfo>
  <tModel tModelKey="">
    <name>International Train Connections Domain Ontology</name>
    <description>Train Itineraries for Ticket Booking Ontology</description>
    <overviewDoc>
      <overviewURL>http://…/doc.xml</overviewURL>
    </overviewDoc>

    <categoryBag>
      <keyedReference tModelKey="DIP Category Classification Scheme"
                      keyName="urn:dip:ontology" keyValue="1.4"/>

      <keyedReference tModelKey="dipBusinessEntity"
                      keyName="urn:dip:nfps:publisher" keyValue="VTA"/>

      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:version" keyValue="1.0"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:coverage" keyValue="Europe"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:language" keyValue="English"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:date" keyValue="20040430"/>
      <keyedReference tModelKey="uddi-org:general_keywords"
                      keyName="urn:dip:nfps:format" keyValue="text"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="international"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="train itineraries"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="ticket booking"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="tickets"/>
      <keyedReference tModelKey="dipSubject"
                      keyName="urn:dip:nfps:subject" keyValue="train"/>
    </categoryBag>

    <identifierBag>
      <keyedReference tModelKey="dipService"
                      keyName="urn:dip:nfps:identifier" keyValue="http://…/tc.flr"/>

      <keyedReference tModelKey="dipBusinessEntity"
                      keyName="urn:dip:businessEntity" keyValue="VTA"/>
      <keyedReference tModelKey="dipBusinessEntity"
                      keyName="urn:dip:nfps:publisher" keyValue="VTA"/>

      <keyedReference tModelKey="dipBusinessEntity"
                      keyName="urn:dip:nfps:creator" keyValue="Max Mustermann"/>

      <keyedReference tModelKey="dipService"
                      keyName="urn:dip:nfps:source" keyValue="http://…/tc.flr"/>
      <keyedReference tModelKey="dipService"
                      keyName="urn:dip:nfps:relation"
                      keyValue="http://…/VTA-Ontology-dateandtime"/>
    </identifierBag>
  </tModel>
</save_tModel>
```
5.7.3 Publish DIP Goal

It is assumed, that Goals must have an owner. Publishing a DIP Goal in the Registry includes the following steps:

1) Define used Categorization Scheme

Table 13: Required Category Bag Entry

<table>
<thead>
<tr>
<th>Category Bag Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization Scheme</td>
</tr>
<tr>
<td>Key Name</td>
</tr>
<tr>
<td>Key Value</td>
</tr>
</tbody>
</table>

Additional Category Bag entries may be added to allow retrieval in non-DIP UDDI usage.

2) Define the owner of the Goal

The owner of a Goal, expected to be the publisher’s name, will be stored in the Identifier Bag to be able to retrieve a Goal by owner.

Table 14: Required Identifier Bag Entry

<table>
<thead>
<tr>
<th>Identifier Bag Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Scheme</td>
</tr>
<tr>
<td>Key Name</td>
</tr>
<tr>
<td>Key Value</td>
</tr>
</tbody>
</table>

Additional Identifier Bag entries may be added to allow retrieval in non-DIP UDDI usage.

3) Create tModel

For each Goal a separate tModel must be created

Table 15: UDDI tModel for ‘DIP Ontology

<table>
<thead>
<tr>
<th>tModel Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Categories:</td>
</tr>
<tr>
<td>Identifiers:</td>
</tr>
<tr>
<td>Overview Document:</td>
</tr>
</tbody>
</table>

Re-publishing a DIP Goal is similar to publishing a new Goal, but requires the Goal key, and may require the preserved Bags.

4) Sample listing

Listing 9: Goal registration sample

```xml
<save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
<authInfo>OBNQxY5jV...eKU0LbNbpOM=</authInfo>
```
5.7.4 Publish DIP Mediator Service

DIP Mediators are handled in a similar way to DIP Web Services. The specification is thus identical to the one described in Section 5.7.1., except that dipWebService has to be replaced with dipMediatorService.

5.8 Retrieval Interface

UDDI has some limitations regarding its query interface. Most of the limitations are due to the use of the OR Bag qualifiers. Solutions to these limitations are not elaborated on in this document.

5.8.1 Retrieval Methods

For retrieval, different methods serve different requirements:

Query

This selects a set of DIP elements restricted by Bags (Identifier Bags, Category Bags, tModel Bags, BusinessEntity). It has to be taken into account that Bags used for retrieval have to use keys and values used during publishing.

A query has the following input:

- authInfo (optional in all known UDDI implementations)
- Category Bag (optional)
- Identifier Bag (optional)
- tModel Bag (optional)
Business Service key (optional)

If no input is provided, from a query point of view all DIP Web Services should be returned, but the actual number depends on the UDDI implementation.

**Batch Retrieval**

Batch Retrieval makes all DIP elements available. For high volumes, a subscription mechanism would be required, which is available in UDDI V3.

The standard UDDI query interface is used. Use of Bags requires knowledge of what properties publishers use in them.

**5.8.2 Retrieval by non-DIP-aware Client**

Today, such a client would retrieve by Category Bag, tModel, partial name, and business provider. The client would select http (or fax, e-mail, etc.) binding. The binding would derive instances and the caller would call the DIP Web Service, unaware of its DIP element properties and binding.

**5.8.3 Retrieve DIP elements**

Retrieving published DIP Goals and Ontologies is straightforward using standard UDDI find functions by any combination of name, business, categories, identifiers taking into account the specifications for Publishing.

The same is true for DIP Web Services (including Mediator Services) unless there is a requirement to search with identifiers (e.g. DIP element properties mapped into Identifier Bags), since UDDI V2 does not allow searching for services by identifiers. The workaround is as follows:

1) **Select Categorization Scheme (mandatory)**

   **Table 16: Required Category Bag Entry**

<table>
<thead>
<tr>
<th>Category Bag Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization Scheme</td>
</tr>
<tr>
<td>&quot;DIP Category Classification Scheme&quot;</td>
</tr>
<tr>
<td>Key Name</td>
</tr>
<tr>
<td>&quot;dipWebService&quot;</td>
</tr>
<tr>
<td>Key Value</td>
</tr>
<tr>
<td>&quot;1.3&quot;</td>
</tr>
</tbody>
</table>

   Additional Category Bag entries may be added.

2) **Select Service owner (optional)**

   **Table 17: Required Identifier Bag Entry**

<table>
<thead>
<tr>
<th>Identifier Bag Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Scheme</td>
</tr>
<tr>
<td>&quot;dipBusinessEntity&quot;</td>
</tr>
<tr>
<td>Key Name</td>
</tr>
<tr>
<td>&quot;businessEntity&quot;</td>
</tr>
<tr>
<td>Key Value</td>
</tr>
<tr>
<td>Name of the Business Entity</td>
</tr>
</tbody>
</table>

3) **Search for tModels**

   UDDI find_tModel function is used, supplying *name*, Category Bag, Identifier Bag as described above.
For each returned tModel, the tModel-detail contains the URL of the document with the DIP element properties.

To find the service, for each tModel-Key, a *find_service* with the Category Bag with exactly one entry as described above must be executed.

### 5.8.4 Sample listings

#### 1) Retrieve Service sample listings

**Listing 10: Find a Service by subject**

```xml
<find_service generic="2.0" maxRows="1000" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference tModelKey="dipSubject"
      keyName="urn:dip:nfps:subject"
      keyValue="tickets"/>
    <keyedReference tModelKey="DIP Category Classification Scheme"
      keyName="urn:dip:service" keyValue="1.3"/>
  </categoryBag>
</find_service>
```

In the following example, all services with source http://www.wsmo.org/resource/dt.flr are queried:

**Listing 11: Find a Service by identifier**

```xml
<find_tModel generic="2.0" maxRows="2" xmlns="urn:uddi-org:api_v2">
  <findQualifiers>
    <findQualifier>exactNameMatch</findQualifier>
  </findQualifiers>

  <identifierBag>
    <keyedReference tModelKey="dipService"
      keyName="urn:dip:nfps:identifier"
      keyValue="http://.../resources/dt.flr"/>
  </identifierBag>

  <categoryBag>
    <keyedReference tModelKey="DIP Category Classification Scheme"
      keyName="urn:dip:service" keyValue="1.3"/>
  </categoryBag>
</find_tModel>
```

#### 2) Retrieve Ontology sample listings

**Listing 12: Find an Ontology by name**

```xml
<find_tModel generic="2.0" maxRows="1024" xmlns="urn:uddi-org:api_v2">
  <findQualifiers>
    <findQualifier>exactNameMatch</findQualifier>
  </findQualifiers>

  <name>Sample Ontology</name>

  <categoryBag>
    <keyedReference tModelKey="DIP Category Classification Scheme"
      keyName="urn:dip:ontology" keyValue="1.4"/>
  </categoryBag>
</find_tModel>
```
Listing 13: Find all Ontologies

```xml
<find_tModel generic="2.0" maxRows="1024" xmlns="urn:uddi-org:api_v2">
    <categoryBag>
        <keyedReference tModelKey="DIP Category Classification Scheme"
                        keyValue="1.4"/>
    </categoryBag>
</find_tModel>
```

Listing 14: Returned summary for all Ontologies

```xml
<tModelList generic="2.0" operator="Sample.UDDI" truncated="false"
            xmlns="urn:uddi-org:api_v2">
    <tModelInfos>
        <tModelInfo tModelKey="uuid:e80f03b6-7585-44d8-bf0d-0b546fa5752d">
            <name>International Train Connections Domain Ontology</name>
        </tModelInfo>
        <tModelInfo tModelKey="uuid:fcc598fa-5bdc-4235-87e2-5708f39f2c3b">
            <name>Travel Ontology</name>
        </tModelInfo>
        <tModelInfo tModelKey="uuid:4baf0272-9153-4bb8-9146-88ed2b877d42">
            <name>Travel Ontology</name>
        </tModelInfo>
        <tModelInfo tModelKey="uuid:00fe4adf-0f36-4afe-8e14-adef11b285cc">
            <name>Another Ontology</name>
        </tModelInfo>
    </tModelInfos>
</tModelList>
```

The request in Listing 13 returns a tModelKey for each matching Ontology. To get details for an Ontology, another query has to be issued using its tModelKey.

Listing 15: Find details for an Ontology

```xml
<get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
    <tModelKey>uuid:e80f03b6-7585-44d8-bf0d-0b546fa5752d</tModelKey>
</get_tModelDetail>
```

Listing 16: Returned detail for an Ontology

```xml
<tModelDetail generic="2.0" operator="Sample.UDDI" truncated="false"
              xmlns="urn:uddi-org:api_v2">
    <tModel tModelKey="uuid:e80f03b6-7585-44d8-bf0d-0b546fa5752d"
            operator="Sample.UDDI" authorizedName="SAMPLE\publisher">
        <name>International Train Connections Domain Ontology</name>
        <description xml:lang="en">International Train Itineraries for Online Ticket Booking Ontology</description>
        <overviewDoc>
            <overviewURL>http://../doc.xml</overviewURL>
        </overviewDoc>
        <identifierBag>
            <keyedReference tModelKey="dipBusinessEntity"
                            keyValue="VTA"/>
            <keyedReference tModelKey="dipBusinessEntity"
                            keyValue="VTA"/>
            <keyedReference tModelKey="dipBusinessEntity"
                            keyValue="DERI"/>
            <keyedReference tModelKey="dipOntology"
                            keyValue="urn:dip:nfps:identifier"/>
        </identifierBag>
    </tModel>
</tModelDetail>
```
3) Retrieve Goal sample listings

**Listing 17: Find all goals**

```xml
<find_tModel generic="2.0" maxRows="1024" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference tModelKey="DIP Category Classification Scheme" keyName="urn:dip:goal" keyValue="1.5"/>
  </categoryBag>
</find_tModel>
```

**Listing 18: Find a goal by name and version**

```xml
<find_tModel generic="2.0" maxRows="1024" xmlns="urn:uddi-org:api_v2">
  <findQualifiers>
    <findQualifier>exactNameMatch</findQualifier>
  </findQualifiers>
  <name> Buy online train ticket </name>
  <categoryBag>
    <keyedReference tModelKey="DIP Category Classification Scheme" keyName="urn:dip:goal" keyValue="1.5"/>
    <keyedReference tModelKey="uddi-org:general_keywords" keyName="urn:dip:nfps:version" keyValue="1.10"/>
  </categoryBag>
</find_tModel>
```
The request in Listing 18 returns a tModelKey for each matching Goal. To get details for a Goal, another query has to be issued using its tModelKey.

**Listing 19: Result for “Find a goal by name and version”**

```xml
<tModelList generic="2.0" operator="sample.UDDI" truncated="false"
xmlns="urn:uddi-org:api_v2">
  <tModelInfos>
    <tModelInfo tModelKey="uuid:e6166754-393d-4f96-a9ec-b2a7620535ab">
      <name>Goal: buying online train ticket</name>
    </tModelInfo>
  </tModelInfos>
</tModelList>
```

**Listing 20: Request for goal details**

```xml
<get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
  <tModelKey>uuid:e6166754-393d-4f96-a9ec-b2a7620535ab</tModelKey>
</get_tModelDetail>
```

**Listing 21: Goal details result**

```xml
<tModelDetail generic="2.0" operator="Sample.UDDI" truncated="false"
xmlns="urn:uddi-org:api_v2">
  <tModel tModelKey="uuid:e6166754-393d-4f96-a9ec-b2a7620535ab"
operator="Herakles.UDDI" authorizedName="HERAKLES\publisher">
    <name>Buy online train ticket</name>
    <identifierBag>
      <keyedReference tModelKey="dipGoal"
 keyValue="http://…/goal.flr"/>
      <keyedReference tModelKey="dipBusinessEntity" keyValue="VTA"/>
      <keyedReference tModelKey="dipBusinessEntity" keyValue="Max Mustermann"/>
    </identifierBag>
    <categoryBag>
      <keyedReference tModelKey="DIP Category Classification
 Scheme" keyValue="1.5"/>
      <keyedReference tModelKey="uddi-org:general_keywords" keyValue="1.10"/>
      <keyedReference tModelKey="uddi-org:general_keywords" keyValue="20040418"/>
    </categoryBag>
  </tModel>
</tModelDetail>
```

**6 CONCLUSION**

Although a customized solution for DIP would provide some benefits, taking UDDI as a starting point for a Registry seems to be a good approach. The main advantage of using UDDI as a first implementation for a DIP Registry is, that a lot of mechanisms needed in a Registry (like access rights, administration, interfaces) are already defined, specified and implemented. Secondly, coexistence with existing Web Services is ensured and existing Web Services can be "upgraded" to DIP by applying DIP element properties.
Mapping of the DIP element properties into the UDDI data model needs some more investigation and should be finalized as soon as the content and format of the properties are fixed and agreed.

7 REFERENCES


