NoTube
*Networks and ontologies for the transformation and unification of broadcasting and the Internet*
FP7 – 231761

---

**D5.3 Discovery, Invocation and Integration of TV Services**

**Coordinator:** D. Liu (OU)

**With contributions from:**
H. Qing Yu (OU), B. Makni (OU), N. Li (OU), C. Pedrinaci (OU), F. Cattaneo (TXT)

**Quality Assessor:** Peter Altendorf
**Quality Controller:** John Domingue

---

**Document Identifier:** NoTube/2011/D5.3/v1.5

**Version:** 1.5

**Date:** 30/10/2011

**State:** Final

**Distribution:** PU
EXECUTIVE SUMMARY

This deliverable describes the discovery, invocation and mediation of NoTube services that are semantically annotated following Linked Services approach presented in D5.2. The implementation of service discovery, invocation and mediation is also guided by Linked Services approach. Thus, its infrastructure and the generic workflow of constructing applications following Linked Services approach are elaborated in this document. In addition, this deliverable presents a case study of Linked Services approach, as well as functional and performance evaluation results.
DOCUMENT INFORMATION

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP7 - 231761</th>
<th>Acronym</th>
<th>NoTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Title</td>
<td>Networks and ontologies for the transformation and unification of broadcasting and the Internet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.notube.tv">http://www.notube.tv</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>Leonhard Maqua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliverable Number</td>
<td>5.3</td>
<td>Title</td>
<td>Discovery, Invocation and Integration of TV Services</td>
</tr>
<tr>
<td>Work Package Number</td>
<td>5</td>
<td>Title</td>
<td>Semantic TV Resource Broker</td>
</tr>
<tr>
<td>Date of Delivery</td>
<td>Contractual</td>
<td>M33</td>
<td>Actual</td>
</tr>
<tr>
<td>Status</td>
<td>Nature</td>
<td>prototype □</td>
<td>report ☑</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>public ☑</td>
<td>consortium □</td>
<td></td>
</tr>
<tr>
<td>Authors (Partner)</td>
<td>OU, TXT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Responsible Author</td>
<td>Name</td>
<td>Dong Liu</td>
<td>E-mail</td>
</tr>
<tr>
<td>Partner</td>
<td>OU, TXT</td>
<td>Phone</td>
<td>+44 (0) 1908-858216</td>
</tr>
<tr>
<td>Abstract (for dissemination)</td>
<td>This deliverable describes the discovery, invocation and mediation of NoTube services that are semantically annotated following the Linked Services approach presented in D5.2. The implementation of service discovery, invocation and mediation is also guided by the Linked Services approach. Thus, its infrastructure and the generic workflow of constructing applications following the Linked Services approach are elaborated in this document. In addition, this deliverable presents a case study of the Linked Services approach, as well as functional and performance evaluation results.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>Linked Services, Service Broker, Semantic Web, SWS, Services, Service Discovery, Service Invocation, Service Integration</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev. No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/09/11</td>
<td>0.1</td>
<td>D. Liu</td>
<td>Preliminary ToC</td>
</tr>
<tr>
<td>03/10/11</td>
<td>0.2</td>
<td>D. Liu, N. Li</td>
<td>Draft content for Section 2 and Section 5</td>
</tr>
<tr>
<td>06/10/11</td>
<td>0.5</td>
<td>D. Liu</td>
<td>Draft content for Section 3, 6 and 7</td>
</tr>
<tr>
<td>12/10/11</td>
<td>0.7</td>
<td>D. Liu, C. Pedrinaci</td>
<td>Section 4</td>
</tr>
<tr>
<td>13/10/11</td>
<td>0.8</td>
<td>F. Cattaneo</td>
<td>Contributions to Section 2</td>
</tr>
<tr>
<td>13/10/11</td>
<td>0.9</td>
<td>B. Makni</td>
<td>Contributions to Section 5</td>
</tr>
<tr>
<td>17/10/11</td>
<td>1.0</td>
<td>D. Liu</td>
<td>Overall refinement</td>
</tr>
<tr>
<td>26/10/11</td>
<td>1.5</td>
<td>D. Liu</td>
<td>Refinement after internal review</td>
</tr>
</tbody>
</table>
# PROJECT CONSORTIUM INFORMATION

<table>
<thead>
<tr>
<th>Participants</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vrije Universiteit Amsterdam</td>
<td>Guus Schreiber</td>
</tr>
<tr>
<td></td>
<td>Phone: +31 20 598 7739/7718</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:schreiber@cs.vu.nl">schreiber@cs.vu.nl</a></td>
</tr>
<tr>
<td>British Broadcasting Corporation</td>
<td>Libby Miller</td>
</tr>
<tr>
<td></td>
<td>Phone: +44 787 65 65 561</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Libby.Miller@bbc.co.uk">Libby.Miller@bbc.co.uk</a></td>
</tr>
<tr>
<td>Pro-netics</td>
<td>Marco Bruni</td>
</tr>
<tr>
<td></td>
<td>Phone: +39 06 45472503</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:marco.bruni@pro-netics.com">marco.bruni@pro-netics.com</a></td>
</tr>
<tr>
<td>Engin Medya Hizmetleri A.S.</td>
<td>Ron van der Heiden</td>
</tr>
<tr>
<td></td>
<td>Phone: +31 6 2003 2006</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:ron@engin.tv">ron@engin.tv</a></td>
</tr>
<tr>
<td>Institut fuer Rundfunktechnik GmbH</td>
<td>Christoph Dosch</td>
</tr>
<tr>
<td></td>
<td>Phone: +49 89 32399 349</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:dosch@irt.de">dosch@irt.de</a></td>
</tr>
<tr>
<td>Ontotext AD</td>
<td>Atanas Kiryakov</td>
</tr>
<tr>
<td></td>
<td>Phone: +35 928 091 565</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:naso@sirma.bg">naso@sirma.bg</a></td>
</tr>
<tr>
<td>Open University</td>
<td>Stefan Dietze</td>
</tr>
<tr>
<td></td>
<td>Phone: +44 1908 858 217</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:s.dietze@open.ac.uk">s.dietze@open.ac.uk</a></td>
</tr>
<tr>
<td>RAI Radiotelevisione Italiana SPA</td>
<td>Alberto Morello</td>
</tr>
<tr>
<td></td>
<td>Phone: +39 011 810 31 07</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:a.morello@rai.it">a.morello@rai.it</a></td>
</tr>
<tr>
<td>Semantic Technology Institute International</td>
<td>Lyndon Nixon</td>
</tr>
<tr>
<td></td>
<td>Phone: +43 1 23 64 002</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:lyndon.nixon@sti2.org">lyndon.nixon@sti2.org</a></td>
</tr>
<tr>
<td>Stoneroos B.V.</td>
<td>Annelies Kaptein</td>
</tr>
<tr>
<td></td>
<td>Phone: +31 35 628 47 22</td>
</tr>
<tr>
<td></td>
<td>Email: annelies.kaptein@stoneroos</td>
</tr>
<tr>
<td>Thomson Video Networks</td>
<td>Raoul Monnier</td>
</tr>
<tr>
<td></td>
<td>Phone: +33 2 99 27 30 57</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:raoul.monnier@thomson-networks.com">raoul.monnier@thomson-networks.com</a></td>
</tr>
<tr>
<td>Polymedia, SpA</td>
<td>Tullio Pirovano</td>
</tr>
<tr>
<td></td>
<td>Phone: +39 02 257711</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:tullio.pirovano@polymedia.it">tullio.pirovano@polymedia.it</a></td>
</tr>
<tr>
<td>KT Corporation</td>
<td>Myoung-Wan Koo</td>
</tr>
<tr>
<td></td>
<td>Phone: +82 2 526 6347</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:mskim@kt.co.kr">mskim@kt.co.kr</a></td>
</tr>
</tbody>
</table>
# Table of Contents

LIST OF FIGURES .................................................................................................................. 7  
LIST OF TABLES .................................................................................................................... 8  
LIST OF ACRONYMS................................................................................................................. 9  

1. INTRODUCTION.................................................................................................................. 10  

2. OVERVIEW OF THE LINKED SERVICES APPROACH.................................................. 11  

3. SMARTLINK 2.0: REFINEMENT AND IMPROVEMENT ................................................. 14  
   3.1. DATA SCHEMA REFINEMENT .................................................................................... 14  
   3.2. ARCHITECTURE REDESIGN .................................................................................... 16  
   3.3. USER INTERFACE ENHANCEMENT ......................................................................... 17  

4. SERVICE DISCOVERY ....................................................................................................... 19  
   4.1. SERVICE SELECTION USING SPARQL ...................................................................... 19  
   4.2. SERVICE DISCOVERY MECHANISMS PROVIDED BY iSERVE .................................... 20  

5. SERVICE INVOCATION AND MEDIATION VIA IRS-III AND OMNIVOKE ................. 23  
   5.1. ARCHITECTURE .......................................................................................................... 23  
   5.2. IMPLEMENTATION ....................................................................................................... 25  

6. BUILDING APPLICATIONS USING LINKED SERVICES ................................................. 28  
   6.1. WORKFLOW AT DESIGN TIME ................................................................................ 28  
   6.2. WORKFLOW AT RUNTIME ...................................................................................... 29  
   6.3. EXTENSIONS TO EXISTING APPLICATIONS ........................................................... 29  

7. EVALUATION RESULTS .................................................................................................... 32  
   7.1. SUGARTUBE: A CASE STUDY OF THE LINKED SERVICES APPROACH .................. 32  
   7.2. FUNCTIONAL EVALUATION ...................................................................................... 36  
   7.3. PERFORMANCE EVALUATION .................................................................................. 40  

8. CONCLUSION .................................................................................................................... 42  

9. REFERENCES ..................................................................................................................... 43
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overall architecture of NoTube service broker</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>NoTube general architecture</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>A partial view of SmartLink NfP schema</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Architecture of SmartLink 2.0</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>SmartLink 2.0 – service editor</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>Adding model references from Watson</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>User interface of service selection and filtering</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>Visualisation of service discovery results</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Architecture of OmniVoke</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>Runtime workflow of NoTube applications built using Linked Services</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>Architecture of SugarTube</td>
<td>32</td>
</tr>
<tr>
<td>12</td>
<td>The SugarTube semantic mining results</td>
<td>35</td>
</tr>
<tr>
<td>13</td>
<td>Relevant videos are linked for better mashup results</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>User interface of SugarTube</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>Search functionalities of SugarTube</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>Semantic-level and data-level mediation as part of SWS discovery, orchestration and invocation</td>
<td>37</td>
</tr>
<tr>
<td>17</td>
<td>Accumulated Zapper EPG service requests per IP</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>Development of EPG requests sent via the broker based on NoTube application requests over time</td>
<td>40</td>
</tr>
<tr>
<td>19</td>
<td>Response time of EPG retrieval services</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Tables
Table 1 NfP schema mapping between SmartLink NfP and ProgrammableWeb ................................. 15
Table 2 Functional evaluation results.................................................................................................... 36
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Deliverable</td>
</tr>
<tr>
<td>EC</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>EPG</td>
<td>Electronic Program Guide</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>GWT</td>
<td>Google Web Toolkit</td>
</tr>
<tr>
<td>IRS</td>
<td>Internet Reasoning Service</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>MSM</td>
<td>Minimal Service Model</td>
</tr>
<tr>
<td>NfP</td>
<td>Non-Functional Property</td>
</tr>
<tr>
<td>OCML</td>
<td>Operational Conceptual Modelling Language</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Schema</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Application</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Term Frequency–Inverse Document Frequency</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modelling Ontology</td>
</tr>
</tbody>
</table>
1. Introduction

This deliverable describes the discovery, invocation and mediation of NoTube services that are semantically annotated following the Linked Services approach presented in D5.2. The implementation of service discovery, invocation and mediation is also guided by the Linked Services approach. Thus, its infrastructure and the generic workflow of constructing applications following the Linked Services approach are elaborated in this document. In addition, this deliverable presents a case study of the Linked Services approach, as well as functional and performance evaluation results.

The rest of this deliverable is divided into seven sections: Section 2 offers an overview of the Linked Services approach and its application in the NoTube project. Section 3 introduces updates and improvements of SmartLink 2.0, an authoring tool for semantic service annotations. Section 4 describes the service discovery mechanisms. Section 5 details the process of service invocation. Section 6 briefly summarises the workflow of building applications following the Linked Data approach. Section 7 reports SugarTube, a case study of developing applications following the proposed Linked Services approach, and the evaluation results. Section 7 also discusses lessons learned from the evaluation. Section 8 concludes this deliverable with a brief summary of the current state of the Linked Services approach and its potential exploitation in future.
2. Overview of the Linked Services Approach

Semantic Web Services (SWS) aims at describing Web services in a machine-understandable way, and then automates the discovery, selection and invocation of Web services. On the other hand, the semantic Web has successfully redefined itself as a Web of Linked (Open) Data (LOD) [1]. Thus, the emerging Linked Services approach [2] exploits the established LOD principles for service description and publication, and is catering for exploiting the complementarity of the Linked Data and services to support the creation of advanced applications for the Web. The distinctive features of the Linked Services approach include 1) a lightweight service model; 2) controllable service discovery and invocation; 3) taking RDF triples as inputs and producing RDF outputs according to domain-specific ontologies. Furthermore, applications built following the Linked Services principles have better flexibility, configurability, extendibility and robustness.

The Linked Services approach is a lightweight, annotation-centric solution to SWS. Compared to comprehensive Web service ontologies like WSMO [3] and OWL-S [4], the core conceptual model of Linked Services, i.e. the Minimal Services Model (MSM), is a set of essentials for formally describing both Web services and RESTful services. In other words, MSM consists of the maximum common denominator between existing formalisms of Web services, which not only caters for interoperability, but also significantly reduces the complexity of ontological descriptions of Web services. In order to explicitly express the semantics of Web services, annotations can be attached to all the components of services, e.g. operations, inputs, outputs, etc. Efforts have also been made to extend MSM so as to cover various context characteristics such as functional classifications and non-functional properties. We present in this document the way of modelling contexts as Linked Data, and define the SmartLink NfP (Non-Functional Property) schema for the service broker developed for NoTube.

In brief, the procedure of the development of an application following the Linked Services methodology includes 1) semantic service annotation and publication, 2) service discovery strategy definition and 3) data schema mapping specification. SmartLink can serve as authoring tool for semantic service annotation. At runtime, applications send the service discovery strategy to SmartLink and iServe to select suitable services, and then OmniVoke [5] and IRS-III [6] perform invocation and execution of those services. Before sending them to the endpoints of services, OmniVoke and IRS-III first extract values from the RDF inputs to prepare the requests. This step is called lowering. When receiving the responses, OmniVoke and IRS-III transform them back into RDF according to the data schema mappings. This is called lifting.

Instead of invoking services directly, applications running on top of Linked Services can interact with a unified interface. SmartLink and iServe, together with OmniVoke and IRS-III, perform service selection and invocation behind that interface. The services to be invoked are determined at runtime. Therefore, applications built following our approach are more flexible. Applications atop the Linked Service are easy to be extended. What developers need to do to extend an existing application includes: 1) semantically describing the new-added services with SmartLink; 2) defining the schema mappings for lowering and lifting and 3) publishing them via iServe. The extensions can be done on the fly, and do not require modification or re-deployment of the existing applications. If some Linked Services are temporally unavailable for reasons like expiration of API keys, connection failures, request timeout, etc., applications can easily filter them out by changing the service discovery strategies. In this way, they are more robust.

---

2. [http://smartlink.open.ac.uk/](http://smartlink.open.ac.uk/)
3. iServe is a registry of Linked Services, which unifies service publication, analysis, and discovery through the use of lightweight semantics. More information can be found at: [http://iserve.kmi.open.ac.uk/](http://iserve.kmi.open.ac.uk/)
Essentially, the semantic service broker is developed following Linked Service principles. As shown in Figure 1, the overall architecture of the semantic TV service broker is a three-tier structure. At the bottom layer, iServe and NoTube NfP repository, a dedicated RDF repository of non-functional properties (NfP) of NoTube services, work as the storage of the semantic service annotations. The NfPs of NoTube services are connected to their behavioral descriptions stored in iServe via rdfs:isDefinedBy. Following those links from SmartLink to iServe, more information about the functionalities of the services can be retrieved. SmartLink operates on top of the storage layer, and exploits the iServe and NoTube NfP repository, and is also interlinked with other Linked Data datasets such as DBpedia.

The tasks related to invocation of Linked Services, which include dereferencing, lowering, grounding, invoking and lifting, are delegated to OmniVoke and IRS-III. When the identifiers of resources on the Web of Data are used as parameters for invoking services, OmniVoke/IRS-III will first attempt to retrieve RDF statements describing those resources, i.e. dereferencing those resources. After that, those RDF statements are lowered to literal values by executing XSPARQL [7] queries or using JRON⁴. Those values are then used to instantiate requests to be sent to the service endpoints.

Grounding refers to the instantiation of service requests, which is the last step of the preparation for the actual invocation of services. After receiving the results in the format of XML, another set of XSPARQL queries or JRON specifications will be executed to transform them back into RDF.

In addition, the service broker has been integrated in the general NoTube architecture depicted in WP6 (see Figure 2). The technical choice for the service integration has been left to the individual use cases, by providing two options:

- Leverage on the Business Process Choreography layer
- Directly invocating the NoTube platform services

The first solution is the most transparent from the application developer perspective since it provides a single entry point for accessing the NoTube services without having to know them all in detail, giving at the same time the possibility to quickly achieve the final goal. It relies upon the Service Broker and thus requires that all the available platform services are properly annotated in advance.

From the WP5 perspective, this approach imposes to collect specific requirements for the requested goals, if not already available, to design the proper orchestration workflow, to perform the semantic annotation of the involved services and to develop the entry point.

---

Direct service invocation, on the other side, is strictly bound to the internal application logic. The developer must have a clear understanding of the specific service input and output parameters and formats as well as the communication protocol adopted. The main drawback here is that there’s no orchestration at all, moreover the integration of new services is no more transparent to the developer.

Figure 2. NoTube general architecture
3. SmartLink 2.0: Refinement and Improvement

This section highlights the achievements of the re-engineering of SmartLink. In short, the new version 2.0 has been released with several new features and abilities. A refined data schema of the NfPs of Web services is structured to improve the applicability and interoperability. The architecture of SmartLink has also been re-designed. The core components have been de-coupled from the GUI and the RESTful interface. Furthermore, SmartLink 2.0 [8] has been equipped with a Linked Data provider, which is in charge of publishing NfPs on the Web of Data, and exposes a SPARQL endpoint. In respect of the user interface, SmartLink 2.0 has been upgraded from a simple Web form to a Rich Internet Application (RIA) implemented using the Google Web Toolkit\(^5\) (GWT), which significantly improves the user experiences.

3.1. Data Schema Refinement

Our previous work dealing with the exploitation of SWS and Linked Services technologies in NoTube project, as described in D5.2, has shown that one of the most established and accepted use cases for Linked Services annotations aims at browsing and searching services in a meaningful way as opposed to automated services discovery and execution. To this end, Linked Services seem of particular use when aiding developers in finding APIs for a given software engineering task.

In this regard, formal specifications turned out to be less important while lightweight service annotations with tags/keywords and classifications played a vital role. Particularly when supporting collaborative annotation of services by a multiplicity of service consumers and developers, formal correctness of the generated data can hardly be enforced and means are required to provide descriptions in a more loose and flexible way. For instance, in many cases, Linked Data resources can be roughly associated with a service, by tagging it with a service category or keyword which might not provide formal enough semantics to facilitate automation of discovery-based execution, but might still be useful to facilitate users in finding appropriate services. For example, an API exposing metadata of resources could be associated with a keyword “metadata” or a reference to http://dbpedia.org/resource/Metadata. However, the current scope of SWS and Linked Services does not provide appropriate facilities to represent such rather lose relationships in an appropriate way but focuses on formal representations of service elements, such as message parts or operations. In that respect, a need for less formal service annotations was observed, to facilitate developers and service consumers to collaboratively annotate services based on Linked Data principles without constraining them by insisting on complete coherence of the provided annotations. Instead of enforcing non-contradictory data, collaborative annotation schemas need to embrace diversity even if that reduces the opportunities for reasoning-based automation.

On a similar note, current service description schemas (e.g., MSM [9], WSMO-Lite [10]) seem to be fundamentally focused on functional properties while not providing sufficient support for NfPs, which would, for instance, allow users to specify licensing schemes, quality of service information or development status descriptions. While some schemas already allow the association of additional service information with particular service instances, the use of dedicated Linked Data vocabularies to further specify NfPs is still underdeveloped. Thus, the SmartLink NfP schema [11], as shown in Figure 3, aims at coming up with a comprehensive data model covering as broad a range of NfPs as possible. On the other hand, in order to meet the need of interoperability, we reuse several existing ontologies and vocabularies, e.g. FOAF\(^6\), CommonTag\(^7\).

---
\(^5\) http://code.google.com/webtoolkit/
\(^6\) http://www.foaf-project.org/
\(^7\) http://commontag.org/
Essentially, SmartLink NfP schema is composed of four parts: social, technical, licensing and QoS. The technical NfPs refer to information about how to interact with the services and include, for instance, the communication protocol (e.g. HTTP and SOAP), data (exchange) format (e.g. XML, RDF and JSON), status (e.g. testing, final, work-in-progress) and authentication model (e.g. HTTP Basic, API Key, OAuth). It is worth noting that technical NfPs do not describe the behaviours of services, but clarify the prerequisites for consumers to make invocation of those services.

The licensing properties indicate the terms and conditions with respect to the usage of individual Web services. As shown in Figure 3, we currently define four concepts for the licensing properties, i.e. service license, data license, usage limits and fees. A service license authorizes and constrains invocation of the service, whereas a data license is for the reuse or repurpose of data generated or provided by the service. Usage limits cover the amount of times of service invocation within a certain time period, or the minimum interval between two times of invocation. Obviously, fees are applicable to non-free services only and refer to the price a consumer needs to pay for consuming a service.

In respect of the quality of Web services, we adopt the model from [12], where the QoS parameters are divided into two classes: objective parameters and subjective parameters. The former are quantitative measures like availability, reliability, throughput and response time, whereas the latter are qualitative measures like user ratings.

<table>
<thead>
<tr>
<th>Table 1 NfP schema mapping between SmartLink NfP and ProgrammableWeb</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SmartLink NfP Schema</strong></td>
</tr>
<tr>
<td>ServiceLicense</td>
</tr>
<tr>
<td>ServiceLicense</td>
</tr>
<tr>
<td>Fee</td>
</tr>
<tr>
<td>Usage Limit</td>
</tr>
<tr>
<td>Authentication Model</td>
</tr>
<tr>
<td>foaf:Organization</td>
</tr>
<tr>
<td>foaf:Company</td>
</tr>
<tr>
<td>foaf:weblog</td>
</tr>
</tbody>
</table>

As stated before, to ensure the applicability and interoperability, we reuse existing ontologies and vocabularies to represent some of the NfPs of Web services. Here, we demonstrate the way of aligning
the proposed schema with schemas adopted by some other Web API repositories. That is important in order to allow interoperability between individual service description repositories or the import of publicly available service NfP metadata into SmartLink 2.0. We use ProgrammableWeb as an example, for the reason that it publishes non-functional properties of more than 4000 Web APIs. Essentially, ProgrammableWeb covers NfPs that are similar to those in our schema, which include Signup and Licensing, Security and Support. Thus, the mappings between our schema and the one of ProgrammableWeb are easy to construct, and parts of the mappings are shown in Table 1.

In addition, both API Status and Mashery offer services of monitoring the QoS parameters of Web APIs. In particular, API Status provides the statistics of the availability and response time of public APIs. Similarly, Mashery also monitors on the availability and response time of another set of Web services. Therefore, not only the data schemas of these two API repositories can be completely mapped to SmartLink NfP schema, but also the data they host can be imported to SmartLink 2.0 and published as Linked Data on the Web.

3.2. Architecture Redesign

Figure 4 depicts the internal structure of the SmartLink 2.0. It runs on top of two different repositories: iServe and NoTube NfP, and provides a joint and integrated overview of the semantic descriptions of services. SmartLink 2.0 is able to merge and consolidate RDF data that spread in distributed repositories by exploiting the references between services descriptions (using rdfs:definedBy) from both data sets but referring to the very same service.

SmartLink 2.0 allows developers to directly annotate existing Web services and APIs, which also includes the annotation of WSMO goals exposed via RESTful APIs which in fact are RESTful services themselves – as MSM service instances.

Figure 4. Architecture of SmartLink 2.0

Being a LOD-compliant environment, one of the core features of SmartLink 2.0 is the capability to associate service descriptions with so-called model references refer to RDF descriptions in external vocabularies defining the semantics of the service or its parts. That way, for instance, a particular service response message can be associated with an external RDF description details and further describes the nature of the response. However, while this feature is useful and even necessary in order to provide meaningful service models, finding appropriate model references across the entire Web of

---

8 http://www.programmableweb.com/apis/directory
9 http://api-status.com/
10 http://developer.mashery.com/status

data is a challenging task. Therefore, SmartLink 2.0 uses established Linked Data APIs – currently the Watson API\(^{11}\) to identify and recommend suitable model references to the user.

Dedicated APIs allow machines and third party applications to interact with SmartLink 2.0, e.g., to submit service instances or to discover and execute services. In addition, the Web application provides a search form which allows querying for particular services. Service matchmaking is being achieved by matching a set of core properties (input, output, keywords), submitting SPARQL queries, and a dedicated set of APIs.

SmartLink 2.0 currently provides mechanisms that enable the export of particular service instances as RDF or human-readable HTML.

3.3. **User Interface Enhancement**

The new user interface of SmartLink 2.0 provides editing and browsing facilities to interact with Linked-Data-based service descriptions. After loading the RDF statements from both iServe and the NoTube NfPs, the editor visualizes the description of a service as shown in Figure 5. The left-hand side of the editor is the tree-based overview of the service, which represents a hierarchy composed of a service, its operations and input/output messages. The right hand side displays more details about the selected element in a form, which essentially includes the semantics, categories, and literal descriptions.

The Web-based service editor of SmartLink 2.0 allows annotation of Web services and Web APIs from scratch, that is, without any pre-existing service documentations such as WSDL or HTML files, as assumed by other annotation tools. SmartLink 2.0 is an open environment accessible to users simply via OpenID\(^{12}\) authentication.

---

3.3. **User Interface Enhancement**

The new user interface of SmartLink 2.0 provides editing and browsing facilities to interact with Linked-Data-based service descriptions. After loading the RDF statements from both iServe and the NoTube NfPs, the editor visualizes the description of a service as shown in Figure 5. The left-hand side of the editor is the tree-based overview of the service, which represents a hierarchy composed of a service, its operations and input/output messages. The right hand side displays more details about the selected element in a form, which essentially includes the semantics, categories, and literal descriptions.

The Web-based service editor of SmartLink 2.0 allows annotation of Web services and Web APIs from scratch, that is, without any pre-existing service documentations such as WSDL or HTML files, as assumed by other annotation tools. SmartLink 2.0 is an open environment accessible to users simply via OpenID\(^{12}\) authentication.

---

\(^{11}\) [http://watson.kmi.open.ac.uk/WatsonWUI/](http://watson.kmi.open.ac.uk/WatsonWUI/)

\(^{12}\) [http://openid.net](http://openid.net)
established Linked Data APIs – currently the Watson API - to identify and recommend suitable model references to the user. Figure 6. shows an example of querying Watson for appropriate model references. In this case, the user tries to find a concept for "Person", and Watson enumerates all the possible choices as a list. All these candidates are extracted from the ontologies known by Watson.

Figure 6. Adding model references from Watson

To persist the changes made to the service description, SmartLink first publishes the functional descriptions as Linked Data. This is done by invoking the RESTful APIs provided by iServe. Then, SmartLink 2.0 will update the NfP repository to save and publish the non-functional properties. Note that SmartLink 2.0 always maintains the links to iServe, so that the functional description can be easily retrieved together with NfPs. In addition, SmartLink 2.0 also offers a simple GUI for filtering services by NfPs (see Figure 7). With this GUI component, the users can easily construct queries without knowledge about SPARQL.

Figure 7. User interface of service selection and filtering
4. Service Discovery

This section presents the services discovery mechanisms provided by SmartLink, as well as by iServe. Given the fact that service descriptions are formalised and published as Linked Data on the Web, the most convenient way of discovering and selecting appropriate services from candidates is executing SPARQL queries against the underlying RDF repositories. Besides that, iServe provides some advanced strategies of services discovery\(^\text{13}\), such as discovery mechanisms based on service inputs/outputs, RDFS classifications, Levenshtein distances and TF-IDF statistics.

4.1. Service Selection using SPARQL

It is simple and flexible to take advantage of SPARQL for querying against the RDF repositories. As stated before, iServe and SmartLink respectively host the functional and non-functional properties of Web services. Therefore, in order to improve the performance of SPARQL query execution and to reduce the overhead of communication with iServe, SmartLink employs a caching mechanism for the service descriptions stored in iServe, i.e. it keeps copies of them locally. When changes being made via the service editor, SmartLink updates the functional service descriptions in iServe, as well as the corresponding links to iServe, to keep its local copies synchronised. With the help of the caching mechanism, both functional descriptions and NfPs can be used together to combine filters of services. An example of complex service filters is shown as follows:

```
PREFIX sawsdl: <http://www.w3.org/ns/sawsdl#>
PREFIX msm: <http://cms-wg.sti2.org/ns/minimal-service-model#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX nfp: <http://www.purl.org/smartlink/service-nfp#>

SELECT ?service WHERE {
  ?service nfp:hasStatus nfp:Final .
  ?service msm:hasOperation ?operation .
  ?operation msm:hasOutput ?output .
  ?output rdf:type msm:MessageContent .
}
```

The SPARQL query above is for finding out all the finalised RESTful services that outcome metadata, i.e. the metadata enrichment services. When receiving the query from the SPARQL endpoint, SmartLink carries out the query execution, and serialises the query results in the format of XML, conforming to the W3C recommendation on this issue\(^\text{14}\). Parts of the execution results of execution the SPARQL query are as follows:


\(^{14}\) [http://www.w3.org/TR/rdf-sparql-XMLres/](http://www.w3.org/TR/rdf-sparql-XMLres/)
To represent the service discovery results in a human-readable way, SmartLink 2.0 is able to visualise the results in a tabular view (see Figure 8).

Figure 8. Visualisation of service discovery results

4.2. Service Discovery Mechanisms provided by iServe

In addition to exposing a SPARQL endpoint on the Web, iServe provides a set of discovery strategies and additional mechanisms supporting the combination and integration of results obtained by diverse techniques to create custom-tailored solutions depending on the current needs. In particular, iServe implements two main types of discovery strategies: discovery with functional taxonomies and RDFS
inferencing and matching of input and output signatures with RDFS inferencing. Additionally we have ported a number of similarity-based approximate matchmaking from iMatcher [13], which was consistently among the best-performing systems in the S3 Contest on Semantic Service Selection15—the reference contest for service matchmaking. In a nutshell these strategies use diverse information retrieval techniques for string similarity analysis over service labels and their input/output signatures.

The discovery mechanisms offered by iServe are available as part of its RESTful API as follows:

http://iserve.kmi.open.ac.uk/data/disco/func-rdfs?class=C1&class=C2

uses RDFS functional classification annotations and returns those services that are related to all the functional categories Ci (which are URIs).

http://iserve.kmi.open.ac.uk/data/disco/io-rdfs?f={and|or}&i=I1&i=I2&o=O1&...

uses ontological annotations of inputs and outputs and returns services for which the client has suitable input data (Ii) and/or (depending on the parameter f for function) which provide the outputs requested by the client (Oi).

http://iserve.kmi.open.ac.uk/data/disco/imatch?strategy=levenshtein&label=L

and

http://iserve.kmi.open.ac.uk/data/disco/imatch?strategy=tfidf&comment=C

performs string similarity matchmaking adopted from iMatcher, returning all services ranked according to the Levenshtein similarity of the service labels with the string L, or according to the TF-IDF weight of the string C in the service descriptions (rdfs:comment).

In the spirit of using Web standards, the RESTful API for discovery represents the discovery results as Atom feeds16, with the entries representing matching services sorted by matching degree. The common representation of discovery results as Atom feeds can be exploited for supporting arbitrary combinations of discovery approaches through list operations on the results of separate discovery queries. iServe currently provides three Atom feed combinators:

- **Union**: the resulting feed contains the entries of all the constituent feeds. For discovery, this is equivalent to the or (disjunction) operator: for discovery queries $D_1 \lor \ldots \lor D_n$, return services that match $D_1 \lor \ldots \lor D_n$.
- **Intersection**: results in a feed with only the entries present in all the constituent feeds. This is equivalent to the and (conjunction) operator for discovery: return services that match $D_1 \land \ldots \land D_n$.
- **Subtraction**: returns the entries of the first feed that are not in any other provided feed. In discovery, this is the and not operator: for queries $D_1 \ldots D_n$, return services that match $D_1 \land \lnot D_2 \land \ldots \land \lnot D_n$.

All these combinators are part of iServe’s RESTful API, and they take feed URIs as parameters. To illustrate the use of the discovery API, including the Atom combinators, the following URI would discover services to geocode US addresses (e.g. look up the geographic location of a postal addresses). The services take US postal address as input, and outcomes geographic locations as outputs.


15 http://www-ags.dfki.uni-sb.de/~klusch/s3/
16 Atom Syndication Format, http://rfc.net/rfc4287.html
And, part of the responses received from iServe, after sending the HTTP request shown above, is presented as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
...
<generator uri="http://iserve.kmi.open.ac.uk/">iServe Atom Union combinator 2010/06/23</generator>
   <updated>2011-10-12T08:37:58.3Z</updated>
...
<entry>
   <id>http://iserve.kmi.open.ac.uk/resource/services/4998b693-2b3f-4cb0-8028-8857c68119e8#wsdl.service(DOTSGeoCoder)</id>
   <title>DOTSGeoCoder</title>
   <updated>2011-10-12T08:36:15.112Z</updated>
   <content>Matching degree: exact</content>
   <link href="http://iserve.kmi.open.ac.uk/resource/services/4998b693-2b3f-4cb0-8028-8857c68119e8#wsdl.service(DOTSGeoCoder)"
    rel="alternate"/>
   <iserve:matchDegree xmlns:iserve="http://iserve.kmi.open.ac.uk/xmlns/"
    num="0">exact</iserve:matchDegree>
   ...  
</entry>
<entry>
   <id>http://iserve.kmi.open.ac.uk/resource/services/c2cc07a1-5a68-4422-958c-4b05e415c84c#wsdl.service(GeoCoderService)</id>
   <title>GeoCoderService</title>
   <updated>2011-10-12T08:36:15.112Z</updated>
   <content>Matching degree: exact</content>
   <link href="http://iserve.kmi.open.ac.uk/resource/services/c2cc07a1-5a68-4422-958c-4b05e415c84c#wsdl.service(GeoCoderService)"
    rel="alternate"/>
   <iserve:matchDegree xmlns:iserve="http://iserve.kmi.open.ac.uk/xmlns/"
    num="0">exact</iserve:matchDegree>
   ...  
</entry>
...
</feed>
5. Service Invocation and Mediation via IRS-III and OmniVoke

Tasks related to service invocation and mediation are achieved by IRS-III and OmniVoke. As described in D5.2, IRS-III is a SWS execution environment, and adopts the WSMO conceptual model of services. IRS-III is able to automatically achieve WSMO goals, but it is not designed for the execution of MSM services. Therefore, OmniVoke is proposed as complement to IRS-III. OmniVoke aims at implementing a unified service invocation engine for heterogeneous services, especially having better support to services prosuming (producing and consuming) Linked Data. It is worth noting that the adoption of OmniVoke is an internal change within the semantic service broker. It has no influence upon the existing services used by the use cases of NoTube project. Some key features of OmniVoke are as follows:

- **Validate invocation request.** In general, an invocation request is issued to an operation of a service. Given a semantic description of the operation to be invoked, OmniVoke should be able to carry out an initial check whether the request is valid, e.g. whether the Web API indeed contains such an operation, by solely looking into its semantic description without communication to the actual service yet.

- **De-capulate invocation request**, figure out what information should be handled in what way, i.e. what information is for OmniVoke’s local use and what information should be passed on to the actual service for invocation use and how. For example, some information in service descriptions can be used to validate requests by OmniVoke and the RDF input data, usually part of the body of the request message for its potentially large size, should be translated into the essential information for carrying out the actual invocation, such as a path parameter, a query string in the address URI, a key-value pair in the HTTP header or an XML message in the HTTP body.

- **Map RDF input data to the expected data format used internally by the service implementation**, e.g. plain literal, XML, JSON etc. The decision may depend on where the input data is grounded in, e.g. the underspecified bits of address URI, message header, message body etc. The mapping process is referred to as “lowering” by the semantic web services community [14].

- **Compose the invocation request to the actual service** using the appropriate URI, method, message header, message body etc. Part of this information can be obtained directly from the service description, such as method, while other parts need to be constructed from the “lowered” input driven by its semantic service description.

- **Invoke the actual service** when a valid request is composed.

- **De-capulate the response message obtained** and figure out what information should be handled in what way. For example, a non-RDF response message body needs to be “lifted” to RDF, as will be described next. Response status and error messages also need to be “lifted” if further usage of that information is anticipated.

- **Map non-RDF data, e.g. XML, JSON in response message body to RDF.** This process is referred to as “lifting”. If a service supports RDF as output format, e.g., Sindice search API or the Geonames search API, no lifting is needed.

5.1. Architecture

With data sources on the Web undergoing a developing trend towards Linked Data, Web APIs, which provide on-the-fly computation of data resources through invocation, need to progress in order to continue playing their roles as Linked Data prosumers when invoked through semantic extensions. Therefore, OmniVoke takes RDF data as input and returns RDF data as response data, thus enabling a seamless integration of Web APIs, as semantic data prosumers [15], into the RDF linked data space. In order to carry out concrete invocations, the envisaged scenario is for applications to issue SPARQL

---

17 [http://technologies.kmi.open.ac.uk/irs](http://technologies.kmi.open.ac.uk/irs)
18 [http://sindice.com/developers/searchapi](http://sindice.com/developers/searchapi)
queries to derive the data required for invoking a particular Web API [16]. Alternatively appropriate
user-interaction interfaces can be provided to allow the user to provide his/her input, which, together
with response data, may be collected into shared data space for further manipulations like inspection,
reuse etc. In latter case, the user is typically confronted with a set of input fields, which need to be
completed in order to invoke the service. Semantic annotations or descriptions of the service can be
attached here to aid the user. Additional information such as comments can be provided to support the
user resolving any potential ambiguities. OmniVoke supports both means of deriving request data. Figure 9 depicts the architecture of OmniVoke.

**Figure 9. Architecture of OmniVoke**

**Request Handler**
The Request Handler is triggered when an invocation request is received. It carries out the tasks of
validating and de-capsulating the invocation request. It acts in correspondence to the set of HTTP
methods or verbs (e.g., GET, PUT, POST, or DELETE). If an action for a given verb is not defined, a
request using such a verb will be answered with HTTP code 405 (Method not allowed). There is no
limit on the number of concurrent activations of such handler.

**Lowering**
The Lowering component undertakes the tasks of “lowering” RDF input data to the format supported
by the actual API. It works by executing the “lowering” scripts designed for each input that requires
lowering and attached to that input in the service description, a mechanism proposed in SAWSDL[20].
However, SASWDL imposes neither restrictions nor prescriptions on the choice of the script language.
Programmatically, XSLT together with SPARQL has been used widely within the community. Lately,
XSPARQL which combines XQuery and SPARQL has been recognized as a more effective language
due to its advantages of avoiding the unnecessary detour of SPARQL query results.

**Request Composer**
The task of the Request Composer is to construct a valid request for invoking the actual Web API
using information given in the service description and the “lowered” input, which now is in the form
supported by the actual Web API.

**Response Handler**
Once the Web API is invoked and the response is returned, the Response Handler is triggered to de-
capsulate the response, i.e. extract the output information (mainly status code, response data), out of
the response header, body etc., and decide whether lifting is required for each output, with the help of
the service description.

Lifting
The Lifting component carries out the execution of “lifting” scripts attached to the output that requires lifting, as annotated in the service description. Similar to “lowering”, lifting scripts can be written in XSPARQL.

Response Composer
Once respective outputs are lifted to RDF, a new response, comprising only RDF data, is constructed by the Response Composer and represented as the final response to the initial invocation request issued to OmniVoke.

5.2. Implementation
OmniVoke exposes its functionality through a Web API that has been made publicly available. Any Web API meant for autonomous invocation via the OmniVoke framework needs to have its interface semantically described using the extended MSM model, which can be done using SmartLink. The semantic descriptions are then published on the semantic Web service publishing platform iServe. In iServe, a Unique Identity (UID) is allocated to every successfully published service description, which will then be used to uniquely identify the URI for a request issued to OmniVoke for the ultimate invocation of the service. Given that a service usually contains not just one operation, an invocation request URI should also indicate which operation is to invoke. Therefore, an invocation request URI is available in the form:

```
http://iserve-dev.kmi.open.ac.uk:8080/RestInvoke/service/{ServiceUID}/operation/{OperationName}/invoke
```

The request is presented to OmniVoke via the POST method because it triggers the creation of resources on the server and thus changes the state of the server. In particular, upon invocation, OmniVoke stores the RDF request data and the RDF response data for clients’ later retrieval or inspection. The underspecified path parameters ServiceUID and OperationName can be obtained from the service descriptions retrieved from iServe.

We use the Lupedia service as an example to illustrate how OmniVoke is implemented and how it works with other components in the world of semantic web service to automate service invocation. In this example, the text2json operation of Lupedia is annotated with the following data schema mappings respectively for lifting and lowering:

```
{
  "prefixes": {
    "lookup": "http://kmi.open.ac.uk/lookup#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  },
  "where": [
    "?lookup rdf:type lookup:Lookup",
    "?lookup lookup:text ?text"
  ],
  "parameters": {
    "lookupText": "text"
  }
}
```

```
{
  "base":"http://kmi.open.ac.uk/lookup#",
  "prefixes":{
    "lookup":"http://kmi.open.ac.uk/lookup#",
```
"mappings": [
    {
        "path": "+",
        "uri": ".instanceUri",
        "properties": [
            {
                "property": "rdf:type",
                "fromData": false,
                "value": "lookup:element",
                "literal": false
            },
            {
                "property": "lookup:Class",
                "fromData": true,
                "value": ".instanceClass",
                "literal": false
            },
            {
                "property": "lookup:Weight",
                "fromData": true,
                "value": ".weight",
                "literal": true
            },
            {
                "property": "lookup:startOffset",
                "fromData": true,
                "value": ".startOffset",
                "literal": true
            },
            {
                "property": "lookup:endOffset",
                "fromData": true,
                "value": ".endOffset",
                "literal": true
            },
            {
                "property": "lookup:predicateUri",
                "fromData": true,
                "value": ".predicateUri",
                "literal": false
            }
        ]
    }
],

When calling Lupedia with the input below:

```json
{
    "__base": "about:blank",
    "__prefixes": {
    },
    "__statements": [
        {
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type": {
                "__iri": "http://kmi.open.ac.uk/lookup#Lookup"
            },
            "http://kmi.open.ac.uk/lookup#text": "%22Enter%20here%20%22public%20"
        }
    ]
}
```
This morning the weather in New York City was foggy, as was the future of HiTech giants IBM and Sun Microsystems. But you will ask - what does the Big Apple have in common with the future of HiTech...? They will be looked up in the DBpedia knowledge base and the enriched text will be returned back. This HTML form can handle texts of up to 7000 characters. Enjoy, Ontotext 22!}

OmniVoke produces the following results. Parts of the invocation results are omitted due to the limitation of space.

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lookup="http://kmi.open.ac.uk/lookup#">
    <lookup:Class rdf:resource="http://dbpedia.org/ontology/MusicalArtist" />
    <lookup:Weight>1</lookup:Weight>
    <lookup:startOffset>34</lookup:startOffset>
    <lookup:endOffset>47</lookup:endOffset>
    <lookup:predicateUri rdf:resource="http://www.w3.org/2000/01/rdf-schema#label" />
  </lookup:element>
  <lookup:element rdf:about="http://dbpedia.org/resource/New_York_City">
    <lookup:Class rdf:resource="http://dbpedia.org/ontology/City" />
    <lookup:Weight>0.99</lookup:Weight>
    <lookup:Weight>0.742499999999999</lookup:Weight>
    <lookup:startOffset>123</lookup:startOffset>
    <lookup:endOffset>136</lookup:endOffset>
    <lookup:predicateUri rdf:resource="http://www.w3.org/2000/01/rdf-schema#label" />
  </lookup:element>
  <lookup:element rdf:about="http://dbpedia.org/resource/IBM">
    <lookup:Class rdf:resource="http://dbpedia.org/ontology/Company" />
    <lookup:Weight>0.98</lookup:Weight>
    <lookup:startOffset>183</lookup:startOffset>
    <lookup:endOffset>186</lookup:endOffset>
    <lookup:predicateUri rdf:resource="http://www.w3.org/2000/01/rdf-schema#label" />
  </lookup:element>
  ...
</lookup:element>
</rdf:RDF>
```
6. Building Applications using Linked Services

This section summarises the workflow of developing applications using Linked Services, which exploits SmartLink, iServe, IRS-III and OmniVoke as the infrastructure. In more detail, semantic service descriptions are created using SmartLink, and stored in iServe in the format of Linked Data. By querying against the SPARQL endpoints of SmartLink or using the service discovery mechanisms provided by iServe, applications select suitable services, and then request IRS-III or OmniVoke to execute them. The benefits brought by using Linked Services as the building blocks of applications are as follows:

- **Dynamics** Applications can determine which services to invoke on the fly.
- **Transparency** The technical details of service selection and invocation are transparent for the other components of an application.
- **Configurability** Maintainers can easily control the behaviours of an application just by revising the strategy of service selection, yet without programming work.
- **Extendibility** To integrate more services offering similar functionalities, developers only need to formally describe those services, and publish them on the Web.
- **Robustness** When a service is temporarily unavailable, a running application can smoothly switch to the alternatives.

The rest of this section details the workflow of developing applications using Linked Services, and explains how the Linked Services infrastructure performs service discovery and execution at runtime. Finally, we illustrate how to make extensions to an existing application in an agile and rapid way, i.e. without hard-coding work or re-deployment.

6.1. Workflow at Design Time

In brief, the proposed approach to construct applications on top of the Linked Services infrastructure includes the following steps:

- **Semantic Services Authoring** This step includes 1) using SmartLink to annotate service descriptions with concepts of the MSM and domain ontologies; 2) publishing them as Linked Data via iServe.

- **Specifying Strategies of Service Selection** This step can be done by either exploiting iServe’s built-in service discovery mechanisms, or writing SPARQL queries to be executed against the RDF dataset of iServe.

- **Defining Lowering and Lifting Schema Mappings** This step outcomes XSPARQL queries or XSLT files for translating RDF triples into parameters used to invoke services, and also for rewriting the invocation results as RDF statements.

Essentially, semantic service authoring is to add annotations to the original documents of service descriptions, so as to make them more understandable for machines. SmartLink has been developed to facilitate annotating both HTML and WSDL files. Although authors can arbitrarily annotate service descriptions, we argue that semantic services will be easier discovered and invoked, if being annotated following principles and patterns shown below.

- Service categories should be attached to services rather than operations or messages. This can simplify service discovery based on functional classifications.
- The addresses and types of HTTP methods, e.g. GET, POST, PUT, etc., should be declared, otherwise operations will not be able to be invoked.
- Information related to groundings of input messages should be provided.
- Lowering schema mappings must be associated with input messages. When an input message has a hierarchical structure, lowering schemas are usually utilised to annotate message parts on the lowest level.
- In principal, lifting schema mappings are for output messages only.
- In many cases, messages are annotated with concepts of domain ontologies, while their sub-parts are annotated with properties of such concepts. This can ensure the alignment of formal semantics of input/output messages and the ontological knowledge. In addition, it also gives hints on writing and understanding the lifting and lowering schema mappings.
In Section 4.1, we have introduced a SPARQL-based service selection, which enables the on-the-fly refinement of the selection strategies. The example in Section 4.1 seeks all the finalised RESTful services for metadata enrichment, i.e. those having the status of “Final”, using “RESTful” as communication method, and returning metadata as outputs.

By means of rewriting that SPARQL query, the applications developed using those metadata enrichment services can behave more adaptively, namely, dynamically choose the services to invoke. Two examples for the typical usage are listed as follows. Developers can create more complex queries to satisfy their own requirements.

```
FILTER (?s != <http://smartlink.open.ac.uk/...#GET-YOUTUBE-TRAILER>)
```

When the service for getting YouTube trailers is now unavailable, this filter can avoid the attempts to invoke it. In this way, it also can meet the requirement for smoothly switching between services at runtime.

```
LIMIT 3
```

The solution sequence modifier LIMIT can restrict the number of services to invoke, so as to reduce the overall response time of the application.

Finally, developers need to define data schema mappings for lowering and lifting. Those schema mappings can be specified in various formats, e.g. XSPARQL, XSLT, JRON. In Section 5.2, we have presented data schema mappings in JRON.

6.2. Workflow at Runtime

At runtime, NoTube applications first send the pre-defined service discovery strategies to SmartLink or iServe, as shown in Figure 10. Then, a set of services is invoked by either OmniVoke or IRS-III. Before sending back to the NoTube applications, the invocation engine merges the results of the service invocation, which deals with issues regarding putting together invocation results from different sources, e.g. eradicating any duplicated items, sorting by specific properties, etc.

Figure 10. Runtime workflow of NoTube applications built using Linked Services

6.3. Extensions to Existing Applications

As stated before, one of the benefits brought by Linked Services is the extendibility. Developers can integrate new semantic services to extend running applications without efforts for the modification of
the source codes. For example, if we want to extend metadata enrichment services with Sindice API\(^\text{23}\), we need to 1) semantically describe it conforming to MSM model; 2) specify the lowering and lifting schema mappings on the basis of the analysis of the sample results of invoking Sindice API. Sindice API has been integrated into SugarTube, a case study of building applications with Linked Services. The semantic descriptions and schema mappings for lowering and lifting are respectively presented as follows.

```sparql
@prefix : <http://iserve.kmi.open.ac.uk/resource/services/4a82b051-7087-40a5-aee7-2caf0432f0ef#> .
@prefix msm: <http://cms-wg.sti2.org/ns/minimal-service-model#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
@prefix sindice: <http://sindice.com/vocab/search#> .

:SindiceService a msm:Service ;
   rdfs:label "Sindice Search Service" ;
   msm:hasOperation :QueryTermOperation .

:QueryTermOperation a msm:Operation ;
   rdfs:comment "Term Search allows you to retrieve documents that are related to keywords and or URIS" ;
   msm:hasInput :QueryTermInput ;
   msm:hasOutput :QueryTermOutput ;
   hr:hasAddress "http://api.sindice.com/v2/search?qt=term&format=rdfxml"^^hr:URITemplate ;
   hr:hasMethod "GET" .

:QueryTermInput a msm:MessageContent ;
   msm:hasPart :q , :page ;
   msm:hasPartTransitive :q , :page .

:q a msm:MessagePart ;
   rdfs:comment "query term" ;
   sawsdl:modelReference sindice:searchTerms .

:page a msm:MessagePart ;
   rdfs:comment "Pages are 1-indexed, so the first page is 1, the second is 2 and so on" ;
   sawsdl:modelReference sindice:page , sindice:startIndex .

:QueryTermOutput a msm:MessageContent ;
```

\(^{23}\) [http://sindice.com/developers/api](http://sindice.com/developers/api)
<requestData>
  <queryParam q="{$term}" page="{$index}"/>
</requestData>

declare namespace rdfs = "http://www.w3.org/2000/01/rdf-schema#";
declare namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
declare namespace search = "http://sindice.com/vocab/search#";
declare namespace dcterms = "http://purl.org/dc/terms/";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace foaf = "http://xmlns.com/foaf/0.1/";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
declare namespace fn = "http://www.w3.org/2005/xpath-functions";

let $doc := doc("OriginalOutputFile")
return
<rdf:RDF>
  { for $entry in $doc//responseData/body/entries
    let $title := $entry/title
    let $link := $entry/link
    return
      <search:Result>
        <dc:title>{data($title)}</dc:title>
        <search:link resource="{data($link)}"/>
      </search:Result>
  }
</rdf:RDF>
7. Evaluation Results

This section presents a case study of developing applications following the Linked Services approach, and also highlights the results of the functional and performance evaluation, as well as lessons learned from the evaluation.

7.1. SugarTube: A Case Study of the Linked Services Approach

We developed SugarTube as a case study for evaluating the Linked Services approach. The following background information aims to provide a broad context for proposing SugarTube: The Open University (OU) is the leading university for providing e-Learning courses in the UK and serves around 200,000 students at all degree levels. In order to support the best quality educational facilities online, the OU has pioneered innovative teaching methods and has produced a wide range of television programmes (e.g. documentaries, history event news and scientific programs) that can serve both students and general audiences. With technologies changing to the digital era, the OU adopts the new digital and multimedia technologies to enable the video resources to be reusable online and accessed by the OU e-Learning platforms. However, these digital videos are still isolated from other educational resources supplied by different providers. Therefore, a mashup browser to accurately search the videos and enrich them with different multimedia resources is proposed.

SugarTube is a mashup application developed to facilitate the usage of the OU’s educational video resources annotated with semantics. We adopt the Linked Service approach to searching videos and exploring their related online multimedia resources through a mashup navigation interface. In SugarTube, the annotations are semantically matched to other annotated educational resources on the Web.

Figure 11. Architecture of SugarTube

As shown in Figure 11, the SugarTube application includes three layers:

- **Interface layer** that interacts with users for specifying the concepts, documents (e.g. lecture notes) or website contents in order to get OU educational video resources and enriched multimedia resources from the Web;
- **Semantic Mining layer** that generates different service discovery and invocation requests based on different types of concept data and user requests;
• **Linked Service layer** that supports dynamic service discovery and invocation in which data is retrieved from different Web services in application domains and with different purposes.

Services are semantically annotated using SmartLink and stored in the iServe. Thus, the services can be dynamically discovered and invoked. There are currently ten different services (see the bottom of Figure 11) used in SugarTube including the Geonames [17] service, the BBC RDF repository [25], OpenLearn [18], YouTube [26], Sindice [19], Zemanta [27], DBpedia [28] and WorldHistory [29].

By using the Linked Service technologies infrastructure, services can be added, modified and deleted whenever they are required. Therefore, the scalability issue is well addressed. The SPARQL query shown below is defined to search for services having an input with two parameters of `sindice:searchTerms` and `sindice:page/sindice:startIndex` (see Sindice search vocabulary specification [30]). Sindice Term-Search service is matched based on the following SPARQL query.

```
SELECT DISTINCT ?o WHERE {
?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> 
?p2 <http://sindice.com/vocab/search#startIndex> .
} LIMIT 1
```

Furthermore, data can be reasoned efficiently by unifying responses from functionally similar services to the RDF format that will then be represented to the interface by applying the same RDF parsing component. The RDF message presented below is sent to OmniVoke to invoke the matched Sindice service.

```
<rdf:RDF xmlns:xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:search = "http://sindice.com/vocab/search#"
  xmlns="http://people.kmi.open.ac.uk/nig/Schema/Sindice/sindice_input.rdf#">
  <search:Query rdf:ID="query0"> 
    <search:searchTerms>Rome</search:searchTerms> 
    <search:page> 
      <rdf:Description search:startIndex="1"/> 
    </search:page> 
  </search:Query> 
</rdf:RDF>
```

25 [http://backstage.bbc.co.uk/data/BbcWebApi](http://backstage.bbc.co.uk/data/BbcWebApi)
28 [http://wiki.dbpedia.org/OnlineAccess](http://wiki.dbpedia.org/OnlineAccess)
29 [http://www.worldhistory.com/api](http://www.worldhistory.com/api)
30 [http://vocab.deri.ie/search#searchTerms](http://vocab.deri.ie/search#searchTerms)
The invocation results which are shown below are derived from the lifting of the matched Sindice service response based on the XSPARQL schema defined by SugarTube developers.

To address the accuracy challenge, we use the Linked Data based search mechanism for semantic rather than plain syntactic matching. To start, one search function is invoked to find or suggest the data of a person (via the DBpedia service), event (via the WorldHistory service) or place (via the GeoName service) from the Linked Open Data cloud [20]. These suggestions allow users to select accurately specified concepts that associate to a URI. Afterwards, the URI is going to be used to search further related learning resources through other Linked Data. For instance, a user enters the place name “Cape Canaveral”. The Geonames searchByName function is invoked consequently. As a result, a service discovery request is sent to the iServe/OmniVoke to search and invoke Geonames services with an XSPARQL customized RDF response. Then, an official place of “Cape Canaveral” is matched in the United States and suggested to the user. If the user commits to the suggested place, the URI http://sws.geonames.org/4149910/ and geo-location information are extracted for searching annotated videos from the OU’s video repository and display the location on the Google map (see Figure 12). As described previously, Linked Data URIs annotate the Open University videos, which is the mechanism we use to match videos to a concept.

For further enriching the learning resources related to the place “Cape Canaveral” and the video, all URIs used for annotating the video (see list on the right side of the video in Figure 13) are extracted as inputs for searching other Linked Data services. For example, related videos are retrieved from Sindice, and OU Linked Data endpoints. Therefore, the learning objects from different resources are linked to each other (see Figure 13). Other learning resources are also linked (e.g. YouTube data).
The user interface of SugarTube consists of five main sections (see Figure 14):

- The search section (top left) that includes two types of searches namely basic concept search and advanced search (see details in Figure 15)
- The event timeline section (bottom left) in which a list of historic events related to the searching concept are displayed and sorted by time
- The OU annotated video displaying section (middle) that allow users to watch videos, view the annotation data and share the video with friends;
- The related knowledge section (top right) that includes the dereferencable data links from the Web (via Sindice), the geo-location map from GeoNames and related learning resource metadata from the OU linked open datasets; and
- The related videos and TV programs section (bottom right) that contains all potentially related videos and TV programs metadata from YouTube, OpenLearn and BBC programs.

The basic concept search separates the concepts into Person, Event, Place and Others. For different types of concept, the different service queries are generated. The Zemanta service is used in the advanced search to support searching videos by automatically analysing documents, highlighting Web contents and pointing to locations on the map. For example, the user can copy and paste the learning content from lecture notes into the text field. Then, all related knowledge concepts are listed, which enables the user to select them for further video searching activities.
7.2. Functional Evaluation

As proposed in D5.2 (see Section 7.1 of D5.2), we are going to analyse the derived functional requirements (EC1-EC5) by means of (a) proof-of-concept implementations and (b) their actual adoption within NoTube.

<table>
<thead>
<tr>
<th>Evaluation criteria</th>
<th>Requirement description</th>
<th>Fulfilled or not</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC1</td>
<td>Abstracting from services implementations based on semantics</td>
<td>Yes</td>
</tr>
<tr>
<td>EC2</td>
<td>Service annotation and tracking facilities for developers</td>
<td>Yes</td>
</tr>
<tr>
<td>EC3</td>
<td>Automated services discovery</td>
<td>Yes</td>
</tr>
<tr>
<td>EC4</td>
<td>Automated services orchestration</td>
<td>Yes</td>
</tr>
<tr>
<td>EC5</td>
<td>Automated services I/O mediation</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**EC1 – Abstracting from services based on semantics:**

We construct MSM and SmartLink NiP to meet this requirement (see Section 3.1 of this deliverable and Section 4 of D5.2). In order to formalise the descriptions of services and to capture their ontological semantics, MSM and SmartLink NiP (see Section 3.1) have been developed. MSM is a simple RDF S ontology that is able to describe (part of) the semantics of both Web services and Web APIs in a common model. SmartLink NiP covers non-functional properties of services. Both MSM and SmartLink NiP are extensible to benefit from the added expressivity of other formalisms.
EC2 – Services annotation and tracking for developers:
SmartLink 2.0 is developed to facilitate authoring semantic service annotations. It now holds 2988 triples, describing the NfPs of 68 services (28 of them are used in the NoTube project). It has been registered as a CKAN package31, and included in the latest version of the Linking Open Data cloud diagram32. As for tracking of developers, the FOAF vocabulary is adopted, and information about 30 developers has been stored in SmartLink in the format of RDF, which includes their name, email address, phone number and history of creating semantic service descriptions with SmartLink.

EC3 - Discovery:
Services are described according to certain criteria, which allow clustering them based on their description parameters, indicating, for instance, the response time of a service, the nature of the exposed content or its language. That way, we are able to create different logical views on the available services. As stated in Section 4, developers are able to execute SPARQL queries against the endpoint of SmartLink or iServe to select suitable services. They can also take advantage of the discovery mechanisms offered by iServe (see Section 4.2).

EC4 – Orchestration:
IRS-III carries out the orchestration of a set of services. IRS-III identifies execution constraints and invokes the selected services as part of a sequence that involves (a) lifting, (b) lowering and (c) consolidation of input/output messages of individual services. Lowering (b) describes the process of transforming an input message (described semantically) into the syntactically correct input message of the underlying service. Lifting (a) represents the process of generating an OCLML instance based on the service output message to enable further processing of service I/O messages. Consolidation (c) refers to the consolidation of individual service output messages into a single response as requested by the requester.

EC5 - Mediation:
Mediation aims at addressing heterogeneities among distinct SWS to support all stages that occur at SWS runtime, namely discovery, orchestration and invocation. We classify the mediation problem into (i) semantic-level and (ii) data-level mediation. Figure 16 illustrates the chronological order of different mediation tasks at SWS runtime. Whereas (i) refers to the resolution of heterogeneities between concurrent semantic representations of services – e.g. by aligning distinct SWS representations – (ii) refers to the mediation between mismatches related to the Web service implementations themselves, i.e. related to the structure, value or format of I/O messages. Hence, semantic-level mediation primarily supports the discovery stage, whereas data-level mediation occurs during orchestration and invocation. Please note that, for the sake of simplification, Figure 16 just depicts mediation between a SWS request and multiple SWS, while leaving aside mediation between different SWS or between different requests.

![Figure 16. Semantic-level and data-level mediation as part of SWS discovery, orchestration and invocation](image)

Below we illustrate the (data) mediation procedure as implemented in the previous example (Italian EPG retrieval). As described in the previous sections, some of our NoTube use case applications

---

31 [http://ckan.net/package/smartlink](http://ckan.net/package/smartlink)
require service responses according to a specific XML schema. Given that our EPG services return data as JSON, the IRS-III is required to apply mediation in terms of lifting from a JSON message into an XML response. For instance, the following service request returns EPG for the Italian channel RAI UNO as part of the service orchestration shown in the previous section.

http://services.notube.tv/epg/datawarehouse.php?service=imdbenrichedperiod&channelid=10027&start=2011-02-03%2000:00&stop=2011-02-03%2023:55

An example of the resulting JSON message is illustrated in the listing below.

```
[{
"uri":"http://purl.org/identifiers/epg/broadcast/10027_2011/02/06_00",
"pid":"10027_2011/02/06_00",
"title":"Euronews",
"start":"2011-02-06:00:00Z",
"stop":"2011-02-06:10:00Z",
"score":0,"votes":0,"source":",
"channelname":"Rai Uno","enrichments":[]},
 {
"uri":"http://purl.org/identifiers/epg/broadcast/10027_2011/02/06_10",
"pid":"10027_2011/02/06_10",
"title":"Aspettando Unomattina - di attualità 06:10",
"start":"2011-02-06:10:00Z",
"stop":"2011-02-06:30:00Z",
"score":0,"votes":0,"source":",
"channelname":"Rai Uno","enrichments":[]}
```

Lifting rules were defined within the IRS-III and associated with the semantic description of the above service in order to allow mediation into the desired XML response. These lifting/lowering rules (a) lift the JSON service response into an ontological instance and (b) generate an XML-response message by lowering the instance based on the desired XML schema. At service execution time, these rules are applied on the fly. For instance, the following goal request performs all the previously discussed tasks (discovery, orchestration, mediation) and finally returns a consolidated XML response.


The result is the following EPG data for RAI1, RAI2, and RAI3, the channels we currently have annotated as "Italian":

```
<metadata>
  <item>
    <title>Telegiornale</title>
    <description></description>
    <url>http://purl.org/identifiers/epg/broadcast/10027_2010_04_19_01_00_00</url>
    <pubDate>2010-04-19T01:00:00Z</pubDate>
    <endDate>2010-04-19T01:30:00Z</endDate>
    <channel>Rai Uno</channel>
    <enrichments>
      <enrichment>
        <eName>Actor</eName>
        <eValue>José Rodríguez Dos Santos</eValue>
        <eUrl>http://www.imdb.com/name/nm0234335/</eUrl>
        <eSource>IMDB</eSource>
      </enrichment>
      <enrichment>
        <eName>Actor</eName>
        <eValue>José Alberto Carvalho</eValue>
        <eUrl>http://www.imdb.com/name/nm1563271/</eUrl>
        <eSource>IMDB</eSource>
      </enrichment>
...
Adoption within NoTube

To underline the suitability and as success measure of the implemented approach, here we present some statistics of usage of the above described goals/services, which show how NoTube partners adopted and integrated the provided goals and, hence, made use of the implemented functionalities (as described above).

Figure 17 shows the accumulated access to the Zapper services from IP addresses across the Web which was obtained from the log files (status: 17/12/2010) of the Zapper Data Warehouse. While there were minor amounts of access from a large number of applications, two IP addresses performed the majority of requests. While there were 7156 requests from 137.108.145.250 which in fact is a server from within the VU Amsterdam network (where also the Zapper services are hosted) which were mainly due to testing purposes, the vast amount of requests (7176) were performed by the service broker (130.37.193.82). That means, that a number of NoTube applications adopted the broker functionalities and integrated the EPG queries via the services brokering provided in WP5.

Figure 17. Accumulated Zapper EPG service requests per IP

Figure 18 illustrates the take up of the Service broker use over time, i.e. a three-month period from 16/08/2010 – 16/11/2010.
7.3. Performance Evaluation

In order to evaluate the performance of the NoTube TV broker, the idea is to run it on top of project-specific orchestration workflows, as commonly requested by WP7.a, b, c by means of goals, then to benchmark the direct invocation of the same services orchestrated by the broker, and finally to compare the obtained results.

EPG retrieval is selected as the scenario for performance evaluation. To setup the benchmark, we developed: (1) reasoning-based services integration via the broker; (2) hard-coded services integration. The former is implemented with IRS-III performing dynamic services selection, rule-based lifting/lowering and keyword filtering, while the latter is a Java stub of case-based service execution without lifting/lowering. Figure 19 shows the response time for the EPG retrieval scenario. It can be seen that reasoning-based service integration via IRS-III is significantly slower, when compared with the Java stubs. The overheads come from data processing (lifting, lowering, filtering) and sequential service invocation. IRS-III can reduce manual efforts of service integration and mediation, but at high costs.

Figure 18. Development of EPG requests sent via the broker based on NoTube application requests over time
Figure 19. Response time of EPG retrieval services
8. Conclusion

In this document, we have described the life-cycle of the Linked Services approach, especially the creation, discovery, invocation and integration of services. SmartLink 2.0 offers the functionalities of browsing and editing. iServe not only works as a public registry of Linked Services, but also provides several service discovery mechanisms. Service invocation and mediation can be done by IRS-III and OmniVoke. In addition, we come up with patterns for service annotations, and the workflow of developing applications using Linked Services. We will continue consolidating the infrastructure of Linked Services, particularly improving the performance. The Linked Services approach will be adopted to build more applications, e.g. the e-Commerce oriented video annotation environment.
9. References


