Ontology based data integration in Life Sciences

Dmitry Repchevskiy
Ontology based data integration in Life Sciences
Ontology based data integration in life sciences

Submitted by Dmitry Repchevskiy in fulfillment of the requirements for the doctoral degree by the University of Barcelona

Supervisor:
Dr. Josep Lluís Gelpí Buchaca
Department of Biochemistry and Molecular Biology
University of Barcelona

Dmitry Repchevskiy
Barcelona Supercomputing Center
National Institute of Bioinformatics
“It is not the consciousness of men that determines their being, but, on the contrary, their social being that determines their consciousness.”

Karl Marx

Without any doubts this thesis wouldn’t be possible without many people that I had a pleasure to work with. If only I tried to enumerate all their invaluable help, all the chats and communications we had, this thesis would require a second volume. First and foremost, I would like to thank my supervisor Josep Lluís Gelpí, who gave me a lot of freedom in defining projects I worked on. Not all of them have been included into this thesis and some of them never reached the end, but the real experience gathered in these years allowed me to go get into this final point. I would also like to thank my colleague José María Fernández from Spanish National Cancer Research Centre (CNIO), who always found a time to discuss technological aspects of my projects and to our entire group just to be with me all these long years. Finally, the greatest thank to my colleagues Romina and Laia whose invaluable support has been so important for me over these years.
As many other science disciplines, Life Sciences operate with an enormous amount of information. The genomic revolution was a big bang in biological and especially genetic data creation. Exponentially growing data had thrown up a bunch of problems with its storage, processing and interoperability. None of these problems could be resolved without a substantial progress in computer technology. On the merge of biology and information technology, a new field, coined as bioinformatics, had arisen. Many heterogeneous data sources continuously generate a huge amount of different types of data. This data comes from clinical studies, micro-array experiments, DNA sequencing, or publications (data mining). In most of the cases, this information has little value without further processing and analysis, including normalization and filtering. To handle this data deluge, many institutions spend a considerable amount of resources to maintain core databases (UniProt, Protein Data Bank, Ensembl, etc.), and are in a continuous search for new approaches in data storage, annotation, and integration. Independently on the origin, biological data requires a structure, the definition of an appropriate storage format, and metadata provision. Sometimes metadata is included into the format, but very often is provided externally in form of annotations. In many cases such annotations, describing a specific knowledge domain, are organized to form an ontology (for instance the Gene Ontology Database) and may be a valuable source of information. Although ontologies are often used to annotate biological data, modern ontology languages provide enough expressibility to structurally describe biological objects that makes them a great choice for biological data interoperability. Another use of ontologies for interoperability purpose is the semantic description of biological services. The power of semantic integration in Life Sciences brought a lot of interest from major bioinformatics institutions that embrace ontologies and more generally all Linked Data technologies as a common platform for biological data integration.
TABLE OF CONTENTS
INTRODUCTION .............................................................................................................. 1

1.1. HETEROGENEOUS DATA INTEGRATION .............................................................. 1
Data formats .................................................................................................................. 4

1.2. XML TECHNOLOGY .............................................................................................. 8

1.3. ONTOLOGIES IN LIFE SCIENCES ........................................................................ 10
OBO format .................................................................................................................. 11

1.4. SEMANTIC WEB ..................................................................................................... 12
1.4.1. Resource Description Framework (RDF) .......................................................... 13
1.4.2. RDF in Attributes (RDFa) ................................................................................ 15
1.4.3. RDF Schema (RDFS) ....................................................................................... 15
1.4.4. SPARQL 1.1 .................................................................................................... 16
1.4.5. OWL 2 Web Ontology Language ..................................................................... 17
1.4.6. The Semantic Web Rule Language (SWRL) ...................................................... 20
1.4.7. Rule Interchange Format (RIF) ........................................................................ 20
1.4.8. Linked Data ..................................................................................................... 21

1.5. WEB SERVICES ...................................................................................................... 22
1.5.1. HTTP ............................................................................................................... 23
1.5.2. SOAP ............................................................................................................... 25
1.5.3. WSDL ............................................................................................................. 25
1.5.4. REST ............................................................................................................... 26
1.5.5. WADL ............................................................................................................. 28

1.6. SEMANTIC WEB SERVICES .................................................................................. 29
1.6.1. OWL-S .......................................................................................................... 31
1.6.2. MOBY-S ........................................................................................................ 32
1.6.3. WSMO ............................................................................................................ 35
1.6.4. WSMO-Lite ................................................................................................... 35
1.6.5. SADI ............................................................................................................... 35
1.6.6. WSDL 2.0 RDF Mapping .............................................................................. 36

OBJECTIVES .................................................................................................................. 37

MATERIALS AND METHODS ....................................................................................... 39
Technological choices ................................................................................................... 41
Semantic Web Services ontology ............................................................................... 41
BioMoby integration libraries ...................................................................................... 41

RESULTS ......................................................................................................................... 43

PART I – BIO MOBY ONTOLOGY MODEL INTEGRATION ........................................ 44
4.1.1. New lightweight Java API for BioMoby Registry access and Web Services execution. 46
4.1.2. BioNemus. Creating SAWSDL bioinformatics services based on BioMoby ontology model ................................................................. 49
4.1.3. SAWSDL-based BioMoby ontology integration ............................ 57

PART II – XML SCHEMA GENERATION FROM OWL 2 ONTOLOGIES ......................... 62
4.2.1. Implementation ........................................................................... 62
4.2.2. OWL 2 Model to XML Schema transformation ............................ 62
4.2.3. Practical applications for bioinformatics Semantic Web Service development ....................................................................................... 69

PART III – ONTOLOGY-BASED SERVICE DESCRIPTION FOR BIOINFORMATICS INTEGRATION ................. 71
4.3.1. BioSWR: Semantic Web services Registry for Bioinformatics .......... 72

PART IV – WEB SERVICES INTEGRATION INTO WORKFLOWS EXECUTION TOOLS ......................... 79
4.4.1. BioSWR Registry integration into the Taverna Workbench ............. 79
4.4.2. Galaxy Gears. Web Services integration into Galaxy workbench ....... 81

DISCUSSION ................................................................................................. 85

CONCLUSION .............................................................................................. 93

BIBLIOGRAPHY .......................................................................................... 97

ANNEX ........................................................................................................ 107
LIST OF FIGURES AND TABLES

“Use a picture. It’s worth a thousand words.”

Arthur Brisbane
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NCBI databases search</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CORBA architecture</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Web Services Architecture</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>PDB file format example</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>ASN.1 and XML interoperability</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>XML standards timeline</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Semantic Web Stack</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>RDF triple</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>RDF graph example</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>SPARQL Update via HTTP protocol</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>The structure of OWL 2</td>
<td>17</td>
</tr>
<tr>
<td>12</td>
<td>Example of different OWL 2 syntaxes</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>RIF dialects</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Web Services stack</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>HTTP Request/Response</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>SOAP Envelope</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>WSDL 1.1 / 2.0 model</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>Evolution towards SWS</td>
<td>29</td>
</tr>
<tr>
<td>19</td>
<td>MobyLite Java API</td>
<td>47</td>
</tr>
<tr>
<td>20</td>
<td>Generated Web Application internals</td>
<td>50</td>
</tr>
<tr>
<td>21</td>
<td>BioNemus functional workflow</td>
<td>52</td>
</tr>
<tr>
<td>22</td>
<td>SOBO ontology in Protegé</td>
<td>69</td>
</tr>
<tr>
<td>23</td>
<td>BioSWR general architecture</td>
<td>73</td>
</tr>
<tr>
<td>24</td>
<td>Example of Semantic Rules definitions</td>
<td>74</td>
</tr>
<tr>
<td>25</td>
<td>SPARQL query example</td>
<td>75</td>
</tr>
<tr>
<td>26</td>
<td>Insert SAWSDL reference via SPARQL UPDATE query</td>
<td>76</td>
</tr>
<tr>
<td>27</td>
<td>Taverna 3.0 BioSWR OSGI plug-in</td>
<td>80</td>
</tr>
<tr>
<td>28</td>
<td>Galaxy Gears Java graphical tool</td>
<td>81</td>
</tr>
<tr>
<td>29</td>
<td>wsdl-generic library architecture</td>
<td>82</td>
</tr>
<tr>
<td>30</td>
<td>Generated Galaxy tool interface</td>
<td>83</td>
</tr>
<tr>
<td>31</td>
<td>Developed frameworks and libraries</td>
<td>91</td>
</tr>
</tbody>
</table>
Table 40. The co
Table 39. WSDL 2.0 and WADL descriptions of the PDBdescription RESTful service
Table 38. BioSWR REST API
Table 37. Java example for SPARQL UPDATE query execution
Table 36. Delete annotation SPARQL query example
Table 35. XML Schema type inheritance breakage example
Table 34. XML Schema type inheritance split example
Table 33. XML Schema type restriction example
Table 32. XML Schema type extension example
Table 31. OWL 2 Datatype representation in XML Schema
Table 30. OWL 2 Properties representation in XML Schema
Table 29. OWL 2 Class representation in XML Schema
Table 28. Tools and libraries used in the projects
Table 27. Correspondence between Ontologies in BioMoby and BioNemus
Table 26. Embedding MOBY
Table 25. web services by the authority
Table 24. BioMoby services description. Example: runTcoffeeEvaluateAlignments service
Table 23. Java getEntryFromPDB RESTful web service execution
Table 22. BLAST web service client
Table 21. Java code example for BLAST web service execution
Table 20. BioNemus commandline parameters
Table 19 XML Schema definition for the MD_Trajectory BioMoby object
Table 18. Correspondence between basic BioMoby objects and BioNemus types
Table 17. Correspondence between BioMoby and BioNemus elements
Table 16. Tools and libraries used in the projects
Table 15. BioMoby message example
Table 14. BioMoby services description. Example: runTcoffeeEvaluateAlignments service
Table 13. WADL description of RCSB getCurrent Web service
Table 12. HTTP methods in RESTful API
Table 11. REST data elements
Table 10. HTTP Methods
Table 9. Phylogenetic character data definition example from EDAM ontology (OWL 2)
Table 8 SPARQL insert query example
Table 7. RDFS vocabulary
Table 6. RDFa Lite properties
Table 5. Phylogenetic character data definition example from EDAM ontology (OBO format)
Table 4. OBO Stanzas
Table 3. BSON Serialization example
Table 2. ASN.1 BER Encoding example
Table 1. XML EXI Encoding example
ACRONYMS AND ABBREVIATIONS

“\( \lim_{n \to \infty} R_n^* = H. \)"

Claude Elwood Shannon
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ASN.1</td>
<td>Abstract Syntax Notation One</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>CDR</td>
<td>Common Data Representation</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DL</td>
<td>Description Logic</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>ECN</td>
<td>Encoding Control Notation</td>
</tr>
<tr>
<td>EXI</td>
<td>Efficient XML Interchange</td>
</tr>
<tr>
<td>FI</td>
<td>Fast Infoset</td>
</tr>
<tr>
<td>GIOP</td>
<td>General Inter-ORB Protocol</td>
</tr>
<tr>
<td>HDT</td>
<td>Header Dictionary Triples</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>ITU-T</td>
<td>ITU Telecommunication Standardization Sector</td>
</tr>
<tr>
<td>JAXB</td>
<td>Java Architecture for XML Binding</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML-Based Web Services</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>LD</td>
<td>Linked Data</td>
</tr>
<tr>
<td>MSM</td>
<td>Minimal Service Model</td>
</tr>
<tr>
<td>OBO</td>
<td>Open Biomedical Ontologies</td>
</tr>
<tr>
<td>OSGi</td>
<td>Open Services Gateway initiative</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>OWL-S</td>
<td>Semantic Markup for Web Services</td>
</tr>
<tr>
<td>PCX</td>
<td>Personal Computer Exchange</td>
</tr>
<tr>
<td>QL</td>
<td>Query Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFa</td>
<td>RDF in Attributes</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Schema</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Applications</td>
</tr>
<tr>
<td>RIF</td>
<td>Rule Interchange Format</td>
</tr>
<tr>
<td>RL</td>
<td>Rule Language</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SADI</td>
<td>Semantic Automated Discovery and Integration</td>
</tr>
<tr>
<td>SAWSDL</td>
<td>Semantic Annotations for WSDL and XML Schema</td>
</tr>
<tr>
<td>SCUFL</td>
<td>Simple Conceptual Unified Flow Language</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SWRL</td>
<td>Semantic Web Rule Language</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XQuery</td>
<td>XML Query</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformations</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
</tr>
<tr>
<td>WebDAV</td>
<td>Web Distributed Authoring and Versioning</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WS-I</td>
<td>Web Services Interoperability Organization</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Service Modeling Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSMO-Lite</td>
<td>Lightweight Semantic Descriptions for Services on the Web</td>
</tr>
<tr>
<td>WSRF</td>
<td>Web Services Resource Framework</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
</tbody>
</table>
INTRODUCTION

“When partners can’t agree
Their dealings come to naught
And trouble is their labor’s only fruit”

Ivan Krylov
1.1. Heterogeneous data integration

The exponentially increasing amount of biological data and its heterogeneity require the usage of an appropriate architecture for management, integration, and interoperability.

Probably the most widespread example of distributed system widely used in biological and medical research is the World Wide Web. Using a system of interlinked hypertext documents, researchers can instantly access many of publicly available databanks and, what is more important, explore biological entities interconnections via hyperlinks. Many organizations developed very powerful Web portals providing an easy access to their biological databases (Figure 1).

However, the human-oriented nature of Web poses a serious limitation for computer data processing and integration. HTML-based Web interfaces are designed for data presentation rather than storage, and its automatic extraction proved cumbersome and error-prone (Neerincx & Leunissen, 2005). An automated access to bioinformatics data and tools is especially important for complex, multi-step analysis that can involve many heterogeneous sources. Direct machine-to-machine interaction requires an architecture that provides functionalities such as transmission protocol, identifiers location, interface description, naming resolution, etc.
One of such architectures was the Common Object Request Broker Architecture (CORBA) (Figure 2), which due to its platform independence represented a clear step forward toward a Service Oriented Architecture (SOA) in bioinformatics (Achard & Barillot, 1997).

The extensive list of supported languages, including C++ and Java, made this architecture quite popular in distributed software development. Although Java platform has its own mechanism for development of distributed systems – Remote Method Invocation (RMI), Java 1.2 included a complete CORBA 2.0 ORB implementation, while RMI was modified to operate over Internet Inter-Orb Protocol (IIOP). Since Java is platform independent, RMI was another technology of choice for distributed development in bioinformatics (Möller, Leser, Fleischmann, & Apweiler, 1999).

Although in the late nineties, CORBA reigned biological data integration projects, Web service technologies quickly surpassed it in popularity. This popularity was generally attributed to the simplicity and provoked a lot of criticism (Gokhale, Kumar, & Sahuguet, 2002) from CORBA advocates. Instead of using a binary protocol, Web services are based on Simple Object Access Protocol (SOAP) protocol. SOAP relies on XML and XML Schema, which are more expressive than Internet Definition Language (IDL) used by CORBA, but less effective in data transmission.

Howbeit, Web services today is a widespread and very complex technology with close to hundred specifications.
Web services can be completely described using Web Services Description Language (WSDL). Although Universal Description Discovery and Integration (UDDI) registry is already the standard way for Web services discovering (Figure 3), many bioinformatics projects were specially oriented to provide an architecture for discovery and distribution of biological data through web services (Bhagat, et al., 2010). Even, in many cases, bioinformatics service providers just publish the WSDL file somewhere on their web site.

As more bioinformatics databases and tools are available in a form of Web services, more complex interactions or workflows are possible. The latter requires another level of abstraction to define Web services cooperation. The de-facto standard for modeling executable workflows - Business Process Execution Language (BPEL) did not become very popular in Life Sciences because of the high degree of bioinformatics services being already in use. Although a lot of work has been done to provide recommendations for bioinformatics Web services development (Pettifer, et al., 2010), the growing popularity of RESTful services led to the situation where many Web services lack WSDL description and, as a consequence, cannot participate in BPEL defined interactions. It should be noted that many of the RESTful Web services could be described via WSDL HTTP Binding, but since most of the Web services tools are oriented to SOAP protocol, this possibility is rarely used. To address these issues, a specially oriented to bioinformatics Web services tool - Taverna (Hull, et al., 2006) was developed at the University of Manchester. Taverna uses its own dataflow-centric workflow language – Simple Conceptual Unified Flow Language (SCUFL) (Oinn, et al., 2006) that allows different types of services to be used within the same workflow.
### Data formats

One of the challenges in heterogeneous data integration is the selection of an appropriate message serialization format. Usually the format is strictly defined by the selected architecture (e.g. CORBA uses General Inter-ORB Protocol (GIOP) protocol, which defines a Common Data Representation (CDR) format for data serialization), while sometimes the choice of the format is more liberal. Serialization formats may be arbitrarily divided into binary based and text based ones.

Historically the choice of the appropriate format was based on the encoded data itself, images were encoded in binary formats like Personal Computer Exchange (PCX), text files using American Standard Code for Information Interchange (ASCII) encoding.

All data formats, including text-based ones, abide some structural rules. Even a simple text file follows natural language grammar. For instance, ASCII-based PDB\(^1\) file format defines its own structural rules (Figure 4).

![Figure 4. PDB file format example](http://www.wwpdb.org/docs.html)

Formats developed to encompass different types of data (text, numerical data, dates, etc.) usually define the supported type system as part of the format specification. Some encoding formats provide a clean separation between structural description and serialization.

Despite the overwhelming number of protocols in use, the number of commonly adopted data formats is quite small. Rapid information growth presents new challenges to provide more efficient encodings for existent formats (Binary JSON, Efficient XML Interchange, etc.).

\(^1\) [http://www.wwpdb.org/docs.html](http://www.wwpdb.org/docs.html)
**Extensible Markup Language (XML)**

XML was introduced in 1998 as a simple human-readable format oriented to the internet interoperability. Being a profile of SGML, XML puts little restrictions on document structure. In 2001 XML Schema 1.0 recommendation was published. The same year, XML Information Set recommendation was published putting a borderline between XML document structure and its serialization format.

Binary XML serialization formats have been proposed by different standardization bodies: Fast Infoset by ITU-T and Efficient XML Interchange (EXI) by W3C (Table 1). Although EXI provides better than ASN.1 BER (see below) encoding compression, latter is more suitable for parsing large documents, providing a node length so parser could skip large chunks of the document.

Nowadays, XML is a backbone technology for the most parts of Web standards.

![XML Schema](example.xml)

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>XML Document</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xs:element name=&quot;Person&quot;&gt;</code></td>
<td><code>&lt;Person&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xs:complexType&gt;</code></td>
<td><code>&lt;name&gt;Socrates&lt;/name&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xs:sequence&gt;</code></td>
<td><code>&lt;birth&gt;-470&lt;/birth&gt;</code></td>
</tr>
<tr>
<td><code>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot;/&gt;</code></td>
<td>&lt;/Person&gt;</td>
</tr>
<tr>
<td><code>&lt;xs:element name=&quot;birth&quot; type=&quot;xs:integer&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td>&lt;/xs:sequence&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/xs:complexType&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/xs:element&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXI encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
</tbody>
</table>

Table 1. XML EXI Encoding example

---


3 [http://www.w3.org/TR/exi/](http://www.w3.org/TR/exi/)
Abstract Syntax Notation One (ASN.1)

ASN.1 was one of the earliest notations to define a variety data types that has been widely adopted especially by telecommunication industry. Its abstract nature does not impose the way how information is encoded, and there are many defined encoding rules (BER, DER, PER, XER, etc.) (Dubuisson, 2000).

ASN.1 provides a high degree of interoperability with XML (Figure 5).

![Figure 5. ASN.1 and XML interoperability](image)

XML Schema may be mapped into ASN.1 notation. Defined in Table 2 “Person” value being encoded via XML Encoding Rules results the same XML document as in Table 1.

ASN.1 is defined by International Telecommunication Union (ITU) and commonly used to describe messages in communication protocols.

<table>
<thead>
<tr>
<th>ASN.1 Notation</th>
<th>BER Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schema</strong></td>
<td>Person ::= SEQUENCE { name UTF8String, birth Integer }</td>
</tr>
<tr>
<td></td>
<td>length = 14</td>
</tr>
<tr>
<td></td>
<td>00 name UTF8String: tag = [0] primitive;</td>
</tr>
<tr>
<td></td>
<td>08 length = 8</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>person Person ::= { name &quot;Socrates&quot;, birth -470 }</td>
</tr>
<tr>
<td></td>
<td>02 length = 2</td>
</tr>
<tr>
<td></td>
<td>FE 2A (-470)</td>
</tr>
</tbody>
</table>

Table 2. ASN.1 BER Encoding example

ASN.1 robustness and effectiveness didn't pass unnoticed by the biomedical community (Ostell, Wheelan, & Kans, 2001) and the format is still in use along other emerged formats.
**JavaScript Object Notation (JSON)**

JSON is a format that became very popular on the WEB 2.0 wave. Natively understood by JavaScript, along with the XML it is widely used in Asynchronous JavaScript requests (AJAX). The simplicity of the format makes this format a popular choice where XML may look ponderous, for instance in Representational State Transfer (REST) Web services oriented to dynamic web applications.

Like early XML specifications, JSON promotes a minimalistic text-based approach for data structure description. In its development JSON runs into the same issues W3C consortium came across a decade ago. The simplicity of JSON left apart such moments as name resolution, document schema, extensive and rich type system, etc. Many of these issues are intended to be solved by different enthusiasts, for instance Binary JSON (BSON)⁴ format (Table 3).

<table>
<thead>
<tr>
<th>JSON</th>
<th>BSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
<td>37</td>
</tr>
<tr>
<td>“Person” : {</td>
<td>0x03 Embedded document</td>
</tr>
<tr>
<td>“name” : “Socrates”,</td>
<td>28</td>
</tr>
<tr>
<td>“birth” : -470 }</td>
<td>UTF-8 String</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
</tr>
<tr>
<td></td>
<td>0x09</td>
</tr>
<tr>
<td></td>
<td>0x10</td>
</tr>
<tr>
<td></td>
<td>0x00</td>
</tr>
</tbody>
</table>

Table 3. BSON Serialization example

Probably the most illustrious example of BSON usage is the open source document database MongoDB⁵.

---

⁴ [http://bsonspec.org/](http://bsonspec.org/)
⁵ [https://www.mongodb.org/](https://www.mongodb.org/)
1.2. XML Technology

Among many data formats XML is indeed the most common one. XML is a subset of Standard Generalized Markup Language (SGML) that was developed as a lightweight alternative for use on the World Wide Web. Unlike Hypertext Markup Language (HTML) which is designed for content visualization, XML is designed to describe data. XML is well suited for automatic processing and is widely used as a data interchange format.

Since its introduction in 1998, XML became a power technology comprised of many specifications.

The term “XML” is usually used to refer to essential set of standards related to XML:

- **Extensible Markup Language (XML)** describes a class of data objects called XML documents.
- **XML Information Set (XML Infoset)** describes an abstract data model of an XML document in terms of a set of information items.
- **Namespaces in XML 1.0** provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.

Other related standards that are important parts of the XML ecosystem:

- **XML Schema** describes XML documents defining constraints on their data model. Unlike Document Type Definitions (DTDs), XML Schema is itself represented in an XML vocabulary.
- **XML Path Language (XPath)** is a language for addressing specific parts of an XML document.

![Figure 6. XML standards timeline](image-url)
• **Extensible Stylesheet Language Transformations (XSLT)** is a declarative language for transforming XML documents. XSLT language uses XPath for XML nodes matching.

• **XML Query (XQuery)** is a functional query language for data stored in XML form. XQuery extends XPath language with so-called “FLWOR” expressions providing similar to SQL functionality for querying XML documents.

XML is generally known as a bandwidth inefficient, human readable, text based format. Fast Infoset (FI) binary encoding format has been defined by the ITU Telecommunication Standardization Sector (ITU-T) and the International Organization for Standardization (ISO) standards bodies as an efficient alternative to the XML document format. Recognizing the need for a compact XML representation, W3C has been developed the Efficient XML Interchange (EXI, Table 2) format which significantly reduces XML document size. Unlike FI, which is based on ASN.1 Encoding Control Notation (ECN), EXI uses built-in datatype representations and employs quite sophisticated technics like channel multiplexing and compression. As a result EXI provides better compression, providing support for many APIs like DOM\(^6\), SAX\(^7\) or StAX\(^8\).

---

\(^6\) [http://www.w3.org/DOM/](http://www.w3.org/DOM/)

\(^7\) [http://www.saxproject.org/](http://www.saxproject.org/)

1.3. Ontologies in Life Sciences

The need to establish a common vocabulary for biological data made ontologies an essential part of Life Sciences. Ontologies are intensively used in medical and health care domains (Stearns, Price, Spackman, & Wang, 2001) (Rector, Rogers, Zanstra, Van Der Haring, & OpenG., 2003). Probably the most well-known example of ontology-based integration initiative in bioinformatics is the Gene Ontology project. Its highly adopted Open Biomedical Ontologies (OBO) file format became very popular in Life Sciences community with many ontologies being developed for a wide range of domains. This popularity led to creation of the OBO Foundry (Smith, et al., 2007) initiative. OBO Foundry ontologies are usually designated as bio-ontologies.

The OBO language is situated somewhere apart from a World Wide Web Consortium (W3C) initiative that promotes a Web Ontology Language (OWL) as a complete set of specifications for authoring ontologies. The domination of OBO language in biological domain quickly disappears, as the community is developing more OWL based ontologies. This trend may be observed at BioPortal (Noy, et al., 2009) open repository of biomedical ontologies9. A lot of efforts are also invested into transition of OBO ontologies to the OWL language (Hoehndorf, Oellrich, Dumontier, Kelso, Rebholz-Schuhmann, & Herre, 2010) (Golbreich, Horridge, Horrocks, Motik, & Shearer, 2007) (Horrocks, 2007).

Ontologies are considered a crucial part of the Semantic Web. Providing access to biological databases via Linked Data endpoints significantly increases the capacity of automatic agents to answer complex biological questions. Data mining tools may perform complex distributed queries involving many heterogeneous biological sources. Semantic Web has received a very positive response from the Life Sciences community, which readily embraces new ways to access data and actively share this knowledge (Garcia Godoy, Lopez-Camacho, Navas-Delgado, & Aldana-Montes, 2013). The integration of heterogeneous biological data via Linked Data technologies (where ontologies play a crucial part), is a major strategy for the European Life Science Infrastructure for Biological Information (ELIXIR) initiative (Crosswell & Thornton, 2012).

9 http://www.bioontology.org/BioPortal
**OBO format**

The OBO ontology language is a description logic language based on a simple flat file format. Structurally OBO document consists of a header and a list of stanzas. Stanzas describe Description Logic (DL) entities such as concept, role and individual (Table 4). Each stanza contains a list of statements in a form of tag-value pairs. Built-in OBO semantics contains an extensive set of tags to describe the entities. It also provides a limited set of XML Schema built-in datatypes.

<table>
<thead>
<tr>
<th>Stanza</th>
<th>OWL 2 analog</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Term]</td>
<td>Class</td>
<td>Terms model real word concepts.</td>
</tr>
<tr>
<td>[Typedef]</td>
<td>ObjectProperty</td>
<td>Typedefs define relations (aka roles, properties, predicates).</td>
</tr>
<tr>
<td>[Instance]</td>
<td>Individual</td>
<td>Instances represent concrete objects that belong to some class.</td>
</tr>
</tbody>
</table>

Table 4. OBO Stanzas

OBO language represents a subset of the OWL concepts sharing many similarities with it. The simplicity of the format made it very popular for ontology development.

While OBO format is quite simple and can be easily edited in any text editor (Table 5), GO Consortium provides biologists with OBO-Edit ontology editing tool (Day-Richter, Harris, Haendel, & Lewis, 2007).

```
[Term]
 id: EDAM_data:0871
 name: Phylogenetic character data
 comment: As defined, this concept would also include molecular sequences, microsatellites, polymorphisms (RAPDs, RFLPs, or AFLPs), restriction sites and fragments
 subset: bioinformatics
 subset: data
 subset: edam
 synonym: "Character" RELATED []
 created_in: "beta12orEarlier"
 def: "Basic character data from which a phylogenetic tree may be generated."
 [http://edamontology.org]
 namespace: data
 is_a: EDAM_data:2523 ! Phylogenetic raw data
```

Table 5. Phylogenetic character data definition example from EDAM ontology (OBO format)
1.4. Semantic Web

Semantic Web\textsuperscript{10} is a W3C initiative to bring heterogeneous data to the Web. Under the Semantic Web umbrella, W3C promotes a large collection of Semantic Web technologies (Figure 7).

In contrast to traditional Web which is based mainly on HTML documents, Semantic Web (sometimes referred as Web 3.0) is based on linked data in a format that can be easily processed by software agents.

The special interest in Semantic Web from Life Sciences community is illustrated by the activity in Semantic Web Health Care and Life Sciences (HCLS) Interest Group\textsuperscript{11}. Semantic Web opens exciting possibilities for biological data integration and interoperability (Neumann, Miller, & Wilbanks, 2004). Improving life science data integration with Semantic Web technologies (Katayama, et al., 2013) is a challenging task in bioinformatics.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{semantic_web_stack.png}
\caption{Semantic Web Stack}
\end{figure}

\textsuperscript{10} http://www.w3.org/standards/semanticweb/
\textsuperscript{11} http://www.w3.org/blog/hcls/
1.4.1. Resource Description Framework (RDF)

RDF is a framework for representing information in the World Wide Web. The information is represented as a collection of triples consisting of a subject, a predicate and an object (Figure 8).

![Figure 8 RDF triple](image)

Predicates denote relationships between nodes (subjects and objects) and are identified by URI references. Nodes may be also represented by the so-called *blank node*, which lacks any intrinsic name but still has a local identifier. Objects may also be *literals* (or constant values).

The collection of triplets forms an RDF graph (Figure 9) which may be serialized in different formats (i.e. Turtle, N3, Manchester, JSON-LD). RDF/XML syntax defines the way to serialize RDF graphs in XML format.

![Figure 9 RDF graph example](image)

The part of WSDL 2.0/RDF ontology that describes getEntryFromPDB BioMoby Web service.

Because RDF/XML\(^{12}\) is the prevalent W3C standard syntax for RDF (Turtle has been recently standardized\(^{13}\)), RDF/XML documents are usually referred as RDF ones.

---

12 [http://www.w3.org/TR/rdf-syntax-grammar/](http://www.w3.org/TR/rdf-syntax-grammar/)
13 [http://www.w3.org/TR/turtle/](http://www.w3.org/TR/turtle/)
Besides the already mentioned text-based formats, there is a great interest in providing more compact Binary RDF Representation\textsuperscript{14}. Header–Dictionary–Triples (HDT) format (Fernández, Martínez-Prieto, Gutiérrez, Polleres, & Arias, 2013) is a binary format that is more compact than other existing RDF serialization formats. HDT separates dictionary from triples and doesn’t require parsing the entire RDF document to access parts of the RDF graph. HDT demonstrates a high level of compressibility and scalability for very large datasets.

RDF defines three predefined build-in types to describe groups of things:

- **rdf:Bag** - A Bag represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members.
- **rdf:Seq** - A Sequence represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant.
- **rdf:Alt** - An Alternative represents a group of resources or literals that are alternatives (typically for a single value of a property).

RDF vocabulary listed in section 5.1 of the specification defines all URI references which are given specific meaning by RDF. These references have defined by the RDF specifications leading substring:

\begin{verbatim}
http://www.w3.org/1999/02/22-rdf-syntax-ns#
\end{verbatim}

The URI corresponds to XML namespace in RDF/XML serialization and conventionally associated with rdf: prefix.

A possibility to represent public bioinformatics databases in RDF format has been successfully explored by the Bio2RDF project (Belleau, Nolin, Tourigny, Rigault, & Morissette, 2008). While Bio2RDF warehouse approach clearly demonstrates benefits of semantic web data integration, the full power of Semantic Web may be achieved by uncovering its distributed nature as more biological databases are exposed in RDF format (Redaschi & Consortium, 2009) (Jupp, et al., 2014).

\textsuperscript{14} \url{http://www.w3.org/Submission/HDT/}
1.4.2. RDF in Attributes (RDFa)

The Web is built around HTML which is designed for information visualization. While HTML pages can contain an enormous amount of information, their automatic processing by software agents is quite complicated. RDFa provides a collection of attributes to express RDF in markup languages such as HTML or XHTML. Embedding RDF-based metadata into (X)HTML pages, improves automatic processing without affecting their visualization.

Oriented to Web authors, RDFa provides simplified RDFa Lite version which consists only of five simple attributes and covers most of the developers’ needs.

<table>
<thead>
<tr>
<th>property</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@prefix</td>
<td>used to assign a short-hand prefix for some vocabulary</td>
</tr>
<tr>
<td>@vocab</td>
<td>specifies default vocabulary to be used</td>
</tr>
<tr>
<td>@typeof</td>
<td>specifies a type of the subject (processed element)</td>
</tr>
<tr>
<td>@property</td>
<td>provides the property (or predicate) for the subject</td>
</tr>
<tr>
<td>@resource</td>
<td>Specifies subject’s identifier (instance id)</td>
</tr>
</tbody>
</table>

Table 6. RDFa Lite properties

1.4.3. RDF Schema (RDFS)

RDF language provides a minimum syntax to define RDF graph data model. The meaning of the model is left undefined unless additional semantics is provided. RDF Schema, abbreviated as RDFS, is a semantic extension of RDF that provides mechanisms for describing groups of related resources and the relationships between these resources. RDFS vocabulary allows to describe simple ontologies via classes and properties.

<table>
<thead>
<tr>
<th>RDFS Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RDFS Classes</strong></td>
</tr>
<tr>
<td>rdfs:Resource, rdfs:Class, rdfs:Literal, rdfs:Datatype,</td>
</tr>
<tr>
<td>rdfs:Container</td>
</tr>
<tr>
<td><strong>RDFS Properties</strong></td>
</tr>
<tr>
<td>rdfs:domain, rdfs:range, rdfs:member, rdfs:subClassOf,</td>
</tr>
<tr>
<td>rdfs:subPropertyOf, rdfs:ContainerMembershipProperty,</td>
</tr>
<tr>
<td>rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy</td>
</tr>
</tbody>
</table>

Table 7. RDFS vocabulary
1.4.4. SPARQL 1.1

SPARQL 1.1 is a set of specifications that facilitate RDF graph content querying and manipulation. SPARQL 1.1 significantly extended the original SPARQL Protocol and RDF Query Language (SPARQL) introducing new features such as Update language, Federated Query, Graph Store HTTP Protocol, etc.

While SPARQL Query Language\(^{15}\) allows RDF data retrieval, SPARQL 1.1 Update\(^{16}\) defines a standard way to update RDF data providing similar to Structured Query Language (SQL) capabilities.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sawsdl: <http://www.w3.org/ns/sawsdl#>
INSERT DATA {
  <urn:lsid:inb.bsc.es#wsdl.interface(describePDB)> sawsdl:modelReference 'http://example.com'
}
```

Table 8 SPARQL insert query example

SPARQL 1.1 Federated Query\(^{17}\) is a SPARQL 1.1 Query Language extension for query execution over explicitly defined SPARQL endpoints. Alternatively to SPARQL 1.1 Update, SPARQL 1.1 introduces the REST-like Graph Store HTTP Protocol\(^{18}\). The protocol uses traditional GET, PUT, POST, and DELETE HTTP terms to manage RDF graphs (Figure 10).

![Figure 10. SPARQL Update via HTTP protocol](http://www.w3.org/TR/sparql11-http-rdf-update/)

---

\(^{15}\) [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)

\(^{16}\) [http://www.w3.org/TR/sparql11-update/](http://www.w3.org/TR/sparql11-update/)

\(^{17}\) [http://www.w3.org/TR/sparql11-federated-query/](http://www.w3.org/TR/sparql11-federated-query/)

1.4.5. OWL 2 Web Ontology Language

The OWL 2 ontology language is a set of specification documents describing its conceptual structure, RDF/XML exchange syntax, semantics and conformance requirements (Figure 11).

OWL 2 language is defined in the OWL 2 Structural Specification document\(^\text{19}\). Any OWL 2 ontology can be represented as an RDF graph\(^\text{20}\). While OWL 2 Structural Specification defines OWL 2 language constructs, the Direct Semantics\(^\text{21}\) specification defines the meaning in terms of Description Logic (DL) concepts. Ontologies interpreted using the Direct Semantics specification are informally called “OWL 2 DL”. Another interpretation is based on RDF-Based Semantics\(^\text{22}\) where meaning is directly assigned to RDF graphs. RDF graphs considered as OWL 2 ontologies are informally called “OWL 2 Full”.

The primary exchange syntax for OWL 2 is RDF/XML, but other concrete syntaxes may also be used (Figure 12).

---

\(^{19}\) [http://www.w3.org/TR/owl2-syntax/](http://www.w3.org/TR/owl2-syntax/)

\(^{20}\) [http://www.w3.org/TR/owl2-mapping-to-rdf/](http://www.w3.org/TR/owl2-mapping-to-rdf/)

\(^{21}\) [http://www.w3.org/TR/owl2-direct-semantics/](http://www.w3.org/TR/owl2-direct-semantics/)

\(^{22}\) [http://www.w3.org/TR/owl2-rdf-based-semantics/](http://www.w3.org/TR/owl2-rdf-based-semantics/)
One of the important characteristics of DL languages is the possibility of implicitly represent knowledge inference via DL reasoners. OWL 2 comes with several profiles that further restrict OWL 2 DL, thus limiting its expressive power for the efficiency of reasoning:

- **OWL 2 EL** profile provides polynomial time reasoning with respect to the size of the ontology and is suitable for very large ontologies. The profile is based on $\mathcal{EL}$ family of description logics that provide only existential quantifications.
- **OWL 2 QL** profile provides similar to conventional relational database systems querying in polynomial time. The profile is aimed at applications that use very large volumes of instance data. Query answering in this profile can be implemented by rewriting queries into a standard relational Query Language (QL).
- **OWL 2 RL** profile provides polynomial time reasoning with respect to the size of the ontology without sacrificing too much expressive power. Reasoning in this profile can be implemented using a standard Rule Language (RL).

Despite several years on from the OWL 2 recommendation, most of the ontologies still use only a fraction of its power (Glimm, Hogan, Krötzsch, & Polleres, 2012). Often ontologies are simply used as a means to provide semantic descriptions that can be used to annotate other resources such as Web services, databases, applications, etc. (Ison, et al., 2013).
The possibility to use DL reasoners stirs interest in OWL language (Jupp, Stevens, & Hoehndorf, 2012), and bio-ontologies are slowly moving towards it (Hastings, et al., 2012).

Table 9. Phylogenetic character data definition example from EDAM ontology (OWL 2)
1.4.6. The Semantic Web Rule Language (SWRL)

Description Logic (DL) languages such as OWL are limited to a formal representation of knowledge and have limited expressiveness that may be extended with rules. SWRL is an expressive OWL-based language that includes a high-level abstract syntax for Horn-like rules. SWRL is based on a combination of the OWL DL dialect of the OWL language with a Rule Markup Language (RuleML) and may be expressed either in OWL XML Presentation Syntax (XML Concrete Syntax) or in OWL RDF/XML exchange syntax (RDF concrete syntax). RDF concrete syntax may be accomplished by applying an XSLT transformation to the OWL XML Presentation syntax. SWRL rule axiom consists of an antecedent (body) and a consequent (head) parts (IF-THEN construct).

With the advent of OWL 2 many SWRL rules may be efficiently expressed as DL axioms (for instance restrictions on datatype properties).

1.4.7. Rule Interchange Format (RIF)

While SWRL was designed as an extension to OWL, there are many other rule languages like N3-Logic (Berners-Lee, Connolly, Kagal, Scharf, & Hendler, 2008), SILK (Grosof, 2009), OntoBrocker (Decker, Erdmann, Fensel, & Studer, 1999), etc. The variability of rule languages creates interoperability and integration difficulties. RIF is a W3C standard for exchanging rules among rule systems and engines. RIF specification describes three dialects that are focused on logic-based and production rule languages.

- **RIF-Core** dialect corresponds to the language of definite Horn rules without function symbols (often called 'Datalog') with standard first-order semantics.
- **RIF-BLD** dialect corresponds to the language of definite Horn rules with equality and standard first-order semantics.
RIF-PRD dialect captures the main aspects of various production rule systems. RIF-PRD semantics is based on OMG Production Rule Representation specification\textsuperscript{23}.

Although RIF dialects were designed primarily for rules interchange, each dialect constitutes a standard rule language and thus may be directly used.

Recognizing that RIF rules should be able to interface with RDF and OWL ontologies, RIF RDF and OWL Compatibility specification is included into RIF specifications set.

1.4.8. Linked Data

Linked Data (LD) is a part of W3C Semantic Web initiative that includes many of described previously technologies and which basic idea is to bring semantic data to the Web. The goal of LD is to consolidate huge amount of semantic data available on the Web via the LD Platform\textsuperscript{24} and other complementary specifications.

The purpose of Linked Data Platform is to establish a set of rules for accessing, updating, creating and deleting RDF resources via HTTP protocol. Note that other specifications already have similar functionality (e.g. SPARQL 1.1 Graph Store HTTP Protocol\textsuperscript{25}).

The interesting feature of LD Platform is a possibility to manage non-RDF data. This feature makes LD Platform an interesting option for non-semantic data integration. Many biological data formats (e.g. PDB, FASTA, PIR, etc.) have no RDF representation, but may be easily referred via LD Platform.

Although LD Platform specification is quite recent, the interest in the platform within Life Science community is very high (Goble, et al., 2013), (Thompson, et al., 2014). The ELIXIR initiative has considered the Linked Data approach as the principal data interoperability strategy in Europe, and real work is already on the way via HORIZON 2020 ELIXIR-EXCELERATE project\textsuperscript{26}.

\textsuperscript{23} http://www.omg.org/spec/PRR/1.0/
\textsuperscript{24} http://www.w3.org/TR/ldp/
\textsuperscript{25} http://www.w3.org/TR/sparql11-http-rdf-update/
\textsuperscript{26} http://cordis.europa.eu/project/rcn/198519_en.html
1.5. Web Services

Web Services is a predominant SOA architecture in the Web.

![Web Services stack](image)

Figure 14. Web Services stack

The platform independence made Web services a preferred choice for many integration projects in bioinformatics.

While Web services are based on many technologies and consists of many components, they usually associated with SOAP protocol (Figure 14). SOAP protocol represents an essence of Web Services message oriented model. WSDL document is used to describe Web services in XML grammar.

Another Web architecture which is gaining popularity in bioinformatics data integration is REST. While REST is based on different conceptual principles than Web services, Web APIs based on REST design often referred as RESTful web services. Especial attention must be given to a difference between RESTful Web APIs and an HTTP protocol REST is based on. Not all HTTP-based APIs are RESTful. In fact, HTTP is also the primary protocol in use for SOAP messages.
1.5.1. HTTP

The RFC-2616\(^{27}\) defines HTTP as an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP is designed as a stateless, request-response protocol for client-server architecture and is a main protocol for the World Wide Web.

The stateless nature of the protocol and available cache-control mechanisms allow to significantly reduce the amount of web traffic and increase overall Web throughput. HTTP defines a set of request methods (or verbs) that have predetermined protocol semantics (Table 10).

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Requests a representation of the specified resource.</td>
</tr>
<tr>
<td>HEAD</td>
<td>Requests headers of the specified resource.</td>
</tr>
<tr>
<td>POST</td>
<td>Requests the web server to accept the data for storage.</td>
</tr>
<tr>
<td>PUT</td>
<td>Requests the web server to store the data.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes the specified resource.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Returns HTTP methods supported for specified resource.</td>
</tr>
<tr>
<td>TRACE</td>
<td>Loop-back the request message.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Converts the request connection to a transparent TCP/IP tunnel.</td>
</tr>
</tbody>
</table>

Table 10. HTTP Methods

Other HTTP verbs may be further defined without breaking existing infrastructure. For instance RFC-5789\(^{28}\) specified the PATCH verb Web


Distributed Authoring and Versioning (WebDAV)\(^{29}\) that extends HTTP with seven new verbs.

All HTTP Response messages include a *status code* which reports whether the operation was successful or no. The first digit of the status code specifies one of five classes of response:

- **1xx**: Informational - Request received, continuing process.
- **2xx**: Success - The action was successfully received, understood, and accepted
- **3xx**: Redirection - Further action must be taken in order to complete the request
- **4xx**: Client Error - The request contains bad syntax or cannot be fulfilled
- **5xx**: Server Error - The server failed to fulfill an apparently valid request

The *status code* is usually followed by a *reason phrase* which is a textual status code interpretation (for instance 404 - “Not Found”, 200 – “OK”, 418 - “I'm a teapot”).

HTTP defines a set of standard headers that provide additional information about the message content. Headers may indicate content media type and encoding. They are also used for content negotiation, cache control or authentication purpose.

The Internet Engineering Task Force (IETF) is recently approved the HTTP 2.0 version of the protocol\(^ {30}\). HTTP 2.0 uses binary message framing and is not compatible with previous versions. However, it keeps HTTP 1.1 semantics unchanged that makes them identical from the application level perspective.


1.5.2. SOAP

SOAP is a lightweight XML-based protocol developed for Web services. Designed to be neutral, SOAP doesn’t impose any particular transport protocol usage and may be used over many protocols such as HTTP, SMTP, TCP, or JMS. To achieve this independence, SOAP message is divided into two parts (Figure 16):

![SOAP Envelope](image)

Figure 16. SOAP Envelope

SOAP Header which contains a message specific part and SOAP Body which contains an actual message payload. The protocol neutrality put an additional complexity, thus gaining a criticism from a REST camp.

1.5.3. WSDL

WSDL 1.1 is an XML format for Web services definition was submitted to the W3C consortium in 2001. Although it was never accepted as a standard it was quickly accepted by industry and is still prevalent format for Web services description. Recognizing its inaccuracy and incompleteness, Web Services Interoperability Organization (WS-I) was formed to improve the specification. In parallel, W3C consortium was working on a second version of WSDL which was to resolve many issues found by WS-I. WSDL 2.0 brought a new component model into a scene and greatly improved the extensibility and interoperability (Figure 17).
WSDL 2.0 deliberately separates the core language from predefined extensions (RPC, SOAP and HTTP bindings). It also separates the component model from the XML infoset which defines WSDL 2.0 syntax. The component model imposes many semantic constraints that cannot be validated using the WSDL 2.0 schema. For instance WSDL 2.0 defines top elements ordering which is not reflected in the WSDL 2.0 schema.

1.5.4. REST

REST is the architectural style developed by W3C Technical Architecture Group (TAG) in parallel with HTTP/1.1 protocol. Being a design pattern, REST principals may be implemented with any application level protocol which provides sufficient means to follow REST principles. These principles are based on a concept of resource which must be uniquely identified by a resource identifier. One of the important constraints of the architecture is resource statelessness. The resource identifier must contain all necessary for the resource location. Identified resources characterize conceptual entities and may be described via various representations. For instance the same image resource may be represented in different image formats. This additional information or media type forms part of representation metadata. Other information such as control data may be also passed by underlying protocol.

Undeniably, HTTP protocol is a primary choice for the REST architecture comprising all necessary elements.
RESTful web services.

These services adopt HTTP verbs to provide resource management and in many cases represent an elegant alternative to traditional SOAP-based ones. While RESTful web services are not limited to standard HTTP methods they usually adopted them for the purpose.

<table>
<thead>
<tr>
<th>HTTP Verb</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Returns a list of items</td>
</tr>
<tr>
<td>PUT</td>
<td>Replace entire collection</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new item</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove entire collection</td>
</tr>
</tbody>
</table>

Table 12. HTTP methods in RESTful API

Because of the simplicity, REST architecture is very popular for bioinformatics Web services development (some examples are RCSB PDB REST API[^31], KEGG REST-like API[^32], ChEMBL Web Services[^33], UniProt[^34]). Along with SOAP services, RESTful Web services may be described via WSDL 2.0 (Guardia, Pires, Vêncio, Malmegrim, & de Farias, 2015). Nowadays, RESTful Web services development has become a routine task for bioinformatics developers.

[^31]: http://www.rcsb.org/pdb/software/rest.do
[^33]: https://www.ebi.ac.uk/chembl/ws
[^34]: http://www.uniprot.org/help/programmatic_access
1.5.5. WADL

WADL is an XML-based language for HTTP-based applications description. In many cases WADL overlaps with WSDL HTTP Binding in the provided functionality, but being specially oriented to the description of RESTful web services, and often considered as much simpler alternative. Like WSDL HTTP Binding, WADL allows defining URI-based parameters, HTTP headers and may include XML Schema definitions (Table 13). While WSDL operates with interfaces and their operations, WADL operates with resources and methods (Takase, Makino, Kawanaka, Ueno1, Ferris, & Ryman, 2008).

```
<wadl:application xmlns:wadl="http://wadl.dev.java.net/2009/02"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <wadl:grammars>
        <xs:schema xmlns:tns="http://www.rcsb.org/pdb/rest/">
            <xs:element name="current" type="current"/>
            <xs:element name="PDB" type="PDB"/>
            <xs:complexType name="current">
                <xs:sequence>
                    <xs:element maxOccurs="unbounded" minOccurs="0" name="PDB" type="PDB"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="PDB">
                <xs:attribute name="structureId" type="xs:string"/>
            </xs:complexType>
        </xs:schema>
    </wadl:grammars>
    <wadl:resources base="http://www.rcsb.org/pdb/rest/">
        <wadl:resource path="getCurrent">
            <wadl:method name="GET">
                <wadl:request>
                    <wadl:param xmlns="" name="PDB" style="query" type="PDB"/>
                </wadl:request>
                <wadl:response>
                    <wadl:representation xmlns="" element="current" mediaType="application/xml"/>
                </wadl:response>
            </wadl:method>
        </wadl:resource>
    </wadl:resources>
</wadl:application>
```

Table 13. WADL description of RCSB getCurrent Web service.
1.6. Semantic Web Services

Semantic Web Services (SWS) initiative is an intention to introduce Semantic Web technologies to the Web Services architecture. While Semantic Web is generally referred as Web of Data, SWS constitutes Web of Applications. SWS promise better interoperability by taking advantage of meaningful, context-based analysis of services functionality.

Given the large number of service providers that offer their Web services to the bioinformatics community, the need to support a certain level of interoperability is an important challenge. This interoperability may be achieved on the syntactic level through a common XML Schema based definitions for biological entities (Kalas, et al., 2010), or providing an additional semantic level that describes these services and may be used for service discovery and matching.

Acknowledging limitations of traditional Web Services many projects provide their own SWS frameworks: MOBY-S, Semantic Markup for Web Services (OWL-S), Web Service Modeling Ontology (WSMO), Lightweight Semantic Descriptions for Services on the Web (WSMO-Lite), or Semantic Automated Discovery and Integration (SADI). The common feature of these projects is the usage of an ontology for service descriptions. On the other hand W3C published Web Services Description Language (WSDL) Version 2.0: RDF Mapping specification providing a possibility to express Web services descriptions in OWL Web Ontology Language (OWL). Unlike other Web Service Description projects, WSDL 2.0 RDF Mapping provides an ontology that directly reflects WSDL 2.0 descriptions, making possible a reverse conversion. WSDL 2.0 RDF mapping is not a standalone specification and defines a limited set of constraints.
the WSDL specification imposes providing a minimalistic ontology to describe Web services. The latter means that validity of Web service description represented in OWL vocabulary cannot be verified through an ontology reasoner.

While mentioned frameworks provide a solid basement for describing relevant aspects of Web services all of them have issues with XML-based type system description. Conventional Web service definition specifies XML as the message interchange format\(^\text{35}\), and although WSDL 2.0 specification anticipates other type system usage\(^\text{36}\), XML Schema is the only type system that it defines. Matching XML Schemas with OWL ontologies is a non-trivial task. The Semantic Annotations for WSDL and XML Schema (SAWSDL) extension defines Schema mapping attributes (\textit{liftingSchemaMapping}, \textit{loweringSchemaMapping}), and while it does not prescribe any particular mapping representation scheme, the Extensible Stylesheet Language Transformations (XSLT) language is generally assumed. Some SWS frameworks consider SAWSDL grounding (Martin, Paolucci, & Wagner, 2007), which in many cases may be seen as an intricacy given that SAWSDL is about to provide semantic annotations to various parts of a WSDL, and in a case of pure semantic representation of WS such annotations may be added directly.

---

\(^{35}\) Definition: A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. [http://www.w3.org/TR/wsa-reqs/](http://www.w3.org/TR/wsa-reqs/)

\(^{36}\) Discussion of Alternative Schema Languages and Type System Support in WSDL 2.0 [http://www.w3.org/TR/wsd120-altschemalangs/](http://www.w3.org/TR/wsd120-altschemalangs/)
1.6.1. OWL-S

OWL-S is a W3C Submission of refined DARPA agent markup language for services (DAML-S). In simple terms OWL-S is OWL ontology to describe SWS. Structurally the ontology is separated in three essential branches of descriptions:

- **Service Profile** - provides general provisional information about the service such as its name, description, or contact information, and may be used to facilitate service discovery.

- **Service Model** - provides detailed information of how to interact with a service. This information is modeled in terms of processes and may describe not only simple, or “atomic” services, but also complex or “composite” ones. The composite services are in essence workflows with a sophisticated control rules defined (Sequence, Split, Split + Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until). Input and output parameters are defined as subclasses of Semantic Web Rule Language (SWRL) variables.

- **Service Grounding** - specifies the details of how to access the service and is consistent with WSDL’s concept of binding. In this way Service Model may be seen as an interface definition while Service Grounding as a concrete protocol definition. OWL-S provides WSDL 1.1 grounding defining properties for the WSDL 1.1 elements. Because OWL-S and WSDL use different type systems, to derive the message part from the atomic process instance, an xsltTransformation property may be used. The latter is similar to SAWSDL lowering schema approach.
1.6.2. MOBY-S

BioMoby (The BioMoby Consortium, et al., 2008) project was indeed a remarkable project in SWS frameworks oriented to bioinformatics. Semantic MOBY (Lord, et al., 2004) project (also known as S-MOBY) made an attempt to bring OWL-DL RDF descriptions for BioMoby web services and finally was integrated as a MOBY-S branch of the BioMoby project. The change of the name was due to the integration with another outstanding initiative – myGrid (Stevens, Robinson, & Goble, 2003). This way myGrid embraced Semantic MOBY and MOBY-S became an implementation of myGrid BioMoby definitions (Wilkinson, Gessler, Farmer, & Stein, 2003) (Wilkinson, Schoof, Ernst, & Haase, 2005).

Creation of BioMoby ontology simplified BioMoby integration with other myGrid projects like Taverna (Kawas, Senger, & Wilkinson, 2006) and extended BioMoby visibility.

The BioMoby ontology consisted of four principal ontologies:

- **Object Ontology** provides structural and semantic descriptions for common biological objects (e.g. “AminoacidSequence”, “AntigenicAnnotation”, etc).
- **Namespace Ontology** defines an underlying source of objects, usually a well-known resource (e.g. “UniProt”, “GO”, “PDB”, etc).
- **Service Ontology** provides exhaustive descriptions for BioMoby Web services execution.
- **Service Types Ontology** provides a hierarchy of functions performed by BioMoby services (“Alignment”, “Retrieval”, etc).
The myGrid ontology already defined many terms that are present in BioMoby. The MOBY-myGrid Service ontology extended the latter providing BioMoby specific terms to effectively describe BioMoby services (Table 14).

```xml
  xmlns:b="http://protege.stanford.edu/plugins/owl/dc/protege-dc.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.inab.org/RESOURCES/MOBY-S/ServiceInstances/inb.bsc.es,runTcoffeeEvaluateAlignments">
    <a:locationURI>http://inb.bsc.es/cgi-bin/mobyServices/dispatchers/asyncDispatcher.cgi</a:locationURI>
    <a:hasServiceDescriptionText>Evaluation of an alignment using Tcoffee.</a:hasServiceDescriptionText>
    <a:hasServiceNameText>runTcoffeeEvaluateAlignments</a:hasServiceNameText>
    <a:providedBy>...</a:providedBy>
    <a:hasOperation>
      <rdf:Description rdf:about="http://www.inab.org/RESOURCES/MOBY-S/ServiceInstances/75b852b15ad35cc89f2cd26b82b2cef">
        <a:hasOperationNameText>runTcoffeeEvaluateAlignments</a:hasOperationNameText>
      </rdf:Description>
    </a:hasOperation>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.inab.org/RESOURCES/MOBY-S/ServiceInstances/5871519782f3424481934981d2773e69">
    <a:hasParameterNameText>alignment</a:hasParameterNameText>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.inab.org/RESOURCES/MOBY-S/ServiceInstances/d16be0769a2ab60a6b0a51c2b391fa1b">
    <a:objectType rdf:resource="urn:lsid:biomoby.org:objectclass:Clustalw_Text:2001-09-21T16-00-00Z"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.inab.org/RESOURCES/MOBY-S/ServiceInstances/e614a99a0d5a7012db8d29067203fe4">
    <a:hasParameterType rdf:resource="http://www.mygrid.org.uk/mygrid-moby-service#simpleParameter"/>
  </rdf:Description>
</rdf:RDF>
```

Table 14. BioMoby services description.
Example: runTcoffeeEvaluateAlignments service
Unlike other ontologies aimed at Web Services description, BioMoby ontology also contained structural information for biological objects serialization.

BioMoby utilizes its own XML-based message format (Table 15) which is loosely defined at the BioMoby documentation.

```xml
<MOBY xmlns="http://www.biomoby.org/moby">
  <mobyContent moby:authority="inb.bsc.es">
    <mobyData queryID="sip_1">
      <Simple articleName="sequence">
        <AminoAcidSequence id="P00807" namespace="UniProt">
          <String articleName="SequenceString">MKKLIFL...</String>
        </AminoAcidSequence>
      </Simple>
    </mobyData>
  </mobyContent>
</MOBY>
```

Table 15. BioMoby message example
1.6.3. WSMO

WSMO is another W3C submission that provides ontological specifications of Semantic Web services. Unlike OWL-S WSMO uses its own ontology language – Web Service Modeling Language (WSML). Given that WSML is specially designed for SWS modeling the difference between them is rather conceptual. WSMO relies on the same WSML components:

- **Ontologies.** Ontologies are domain specific ontologies that are in use by other WSMO components. Any WSMO component may be extended by non-functional properties based on Dublin Core Metadata Element Set.
- **Goals.** Goals describe desired functionality for the service. It is similar to service interface but from the user perspective and does not specify any preconditions (input).
- **Web Services.** Web Services describes the Web service functionality (capability) and interactions (interfaces). Interfaces expose their internal functionality in a form of either choreography or orchestration. Choreography provides all necessary details for the client-service interaction, similar to OWL-S AtomicProcess. Orchestration is the pattern of interactions with other Web services in order to achieve the goal and is similar to OWL-S CompositeProcess.
- **Mediators.** Mediators are the core concept to resolve incompatibilities on the data, process and protocol level providing appropriate conversions.

1.6.4. WSMO-Lite

WSMO-Lite (Vitvar, Kopecký, Viskova, & Fensel, 2008) is yet another W3C submission based on the Minimal Service Model (MSM) (Pedrinaci, Kopecký, Maleshkova, Liu, Li, & Domingue, 2011). Unlike WSMO which is based on WSML language, WSMO-Lite is based on OWL. MSM already covers WSDL 1.1 essential descriptions and WSMO-Lite further extends the ontology with some WSMO concepts such as conditions and effects (capabilities) and SAWSDL properties. SAWSDL Schema mapping provides a link between semantic type system and XML Schema model.

1.6.5. SADI

SADI (Wilkinson, Vandervalk, & McCarthy, 2011) is not positioned as a standard and consists of a number of recommendations for semantic service description. SADI does not define any service description ontology, but the framework itself uses MOBY-myGrid one. It also does not define any type
system proposing direct RDF data usage for service input and output. This way SADI services may be seen as RESTful Web services that use RDF as a message interchange format. Unlike XML-based protocols, that usually require strict syntax based on XML Schema, the validity of the SADI message may be dynamically determined by the semantic reasoner.

### 1.6.6. WSDL 2.0 RDF Mapping

WSDL 2.0 RDF Mapping (Kopecký, 2006) is the only W3C recommendation\(^{37}\) to represent Web services as OWL ontology. The ontology follows WSDL 2.0 component designator specification for IRI-references\(^ {38}\). WSDL 2.0 component local names are represented as a literal value of rdfs:label property. The ontology does not strictly follow the WSDL 2.0 component model in places where OWL expressiveness is more appropriate. For instance WSDL 2.0 extension mechanism is better represented through OWL inheritance. WSDL 2.0 ontology provides mapping for predefined WSDL 2.0 extensions defined in WSDL 2.0 Adjuncts\(^ {39}\).

The ontology does not enforce any structural or logical restrictions over components, and may not be used for Web Services validation.

As other ontologies targeting SWS description, this ontology faces a problem with XML Schema type system representation, providing only Qualified Names of element declarations. It should be noted that neither OWL nor OWL 2 support xs:QName datatype and that WSDL 2.0 ontology uses its own wsdl:QName class instead.

---

\(^{37}\) [http://www.w3.org/TR/wsdl20-rdf/](http://www.w3.org/TR/wsdl20-rdf/)

\(^{38}\) [http://www.w3.org/TR/wsdl20/#wsdl-iri-references](http://www.w3.org/TR/wsdl20/#wsdl-iri-references)

\(^{39}\) [http://www.w3.org/TR/wsdl20-adjuncts/](http://www.w3.org/TR/wsdl20-adjuncts/)
OBJECTIVES

“A problem well stated is a problem half solved.”

Charles Ketterin

The need in standard approaches for biological data integration is especially important in a context of ever increasing number of biological databases (Galperin & Fernandez-Suarez, 2011). Ontologies occupy a paramount position in Life Sciences providing an ample coverage for many biological domains. On the other hand, a lot of biological data cannot be statically described and is a result of some software function. As applied to Web Architecture such functions are referred to as Web services. Providing semantic descriptions for biological methods appears to be as important as providing semantic descriptors for biological data. While a lot of frameworks have been proposed as a solution, none of them became a standard and taking into account the fast pace of the Semantic Web standardization process, this area of research constitutes a broad field of investigation. The main objective is to investigate emerging W3C standards in Semantic Web and their applicability to data integration in bioinformatics. Three goals addressed by the thesis are:

- Provide a clear path for bioinformatics services development based on ontologically defined data.
- Provide a transition path of the already established BioMoby platform to the W3C standard-based solutions.
- Provide a practical and standard-based solution for the description of bioinformatics methods based on ontological languages.

In consistency with objectives, thesis results are divided into several parts:

- OWL 2 to XML Schema conversion tool to facilitate bioinformatics web services creation (with an example of a creation of semantically annotated sequence alignment Web service based on OWL 2 ontology)
- Two different approaches for BioMoby web services integration: Automatic web services proxy generation based
on BioMoby ontology and BioMoby ontology integration into WSDL 2.0 descriptors.

- Semantic Web Services Registry based on OWL representation of WSDL 2.0 descriptions.
- The integration of the developed Registry with Taverna Workflow management and enactment tool.
“A cudgel is the intellectual property of barbarians”

Eugene Kascheev
Technological choices

Previously formulated goals require a thorough analysis of existent approaches in semantic data integration in bioinformatics. The analysis includes an examination of requirements for the bioinformatics interoperability with a special accent on compatibility with existent technological solutions. The latter is especially important in a light of practical usage of developed framework. Creation of the semantic framework also requires an evaluation of available libraries for the Semantic Web initiative and probably software development where existent tools are absent or unsuitable.

Semantic Web Services ontology

From a variety of semantic Web service description languages W3C Web Services Description Language (WSDL) Version 2.0: RDF Mapping specification has been chosen as a basement for the project. The choice is explained by its direct WSDL coupling where both representations can be used interchangeably. The latter is especially important given that many of bioinformatics Web services already have their WSDL definition.

WSDL 2.0 component model is used as a core for the Semantic Web Registry. Although WSDL 2.0 component model is quite different from the WSDL 1.1 one, the conversion is still possible and was already anticipated by the W3C40. WSDL 2.0 also provides better than WSDL 1.1 HTTP-based Web services applications description, what facilitates RESTful Web services description.

BioMoby integration libraries

Although Java BioMoby API (jMoby) provides all the functionality to search and execute BioMoby Web services, it doesn’t provide a consistent model to describe them. Every BioMoby Registry Web service requires different parameters for the execution. The MobyCore and MobyCentral libraries are based on the same JAXB-based model that can be easily serialized to the XML description file. JAXB API is also an integral part of the standard JAX-WS API that allowed to tremendously reduce libraries size.

40 http://www.w3.org/2006/02/WSDLConvert.html
<table>
<thead>
<tr>
<th>Java Servers</th>
<th>Standards</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>jBoss AS 7.2.1</td>
<td>Java EE 7 Server</td>
<td>BioSWR</td>
</tr>
<tr>
<td>Apache Tomcat 6 +</td>
<td>Java Servlet Container + Jersey</td>
<td>BioNemus</td>
</tr>
<tr>
<td>GlassFish METRO +</td>
<td>Java API for XML-Based Web Services 2.0 +</td>
<td></td>
</tr>
<tr>
<td>Jersey</td>
<td>Java API for RESTful Services</td>
<td></td>
</tr>
</tbody>
</table>

**Web Frameworks**

| RichFaces 4.5         | JavaServer Faces 2.1 UI component framework   | BioSWR    |

**Database Servers**

| MySQL 5.1 Server      | SQL Database Server                           | BioSWR    |

**Semantic Libraries**

| The OWL API           | OWL 2 Web Ontology Language                  | BioSWR, OWL2XS |
| Hermit OWL Reasoner   |                                               | BioSWR, OWL2XS |
| Sesame                | SPARQL 1.1                                    | BioSWR      |

**WSDL/XML Schema parsers**

| WSDL4j                | WSDL Version 1.1                              | BioSWR, Galaxy Gears |
| Apache Woden          | WSDL Version 2.0                              | Galaxy Gears        |
| Apache XmiSchema 2.1  | XML Schema Language 1.1                       | BioSWR, BioNemus, OWL2XS, Galaxy Gears |

Table 16. Tools and libraries used in the projects
“Nothing happens until something moves.”

Albert Einstein
Part I – BioMoby ontology model integration

Emerged a decade ago, web services have been presented as the answer to rationalize the landscape of modern bioinformatics. Web services could be found through the use of generally available catalogues and the strict specification of data formats makes possible to build workflows of compatible services and perform complex bioinformatics analyses. This would draw a scenario where non-experts could make use of bioinformatics as a routine tool without a deep knowledge of the techniques involved. Besides, the programmatic nature of Web services allows performing genome-wide analyses that are not feasible through classical web applications. Despite of this ideal perspective, present Web services lack the expected acceptance, no common specification adopted by service providers and significant compatibility issues persist.

Initiated in 2001, and with the first stable version published in 2008, BioMoby project was one of the earliest intentions to create a Web services platform for bioinformatics. Indeed, it became a very popular open source framework with thousands of services developed by many organizations. The distinct feature of BioMoby was a semantic layer in the definition of data types that allows non-experts to understand the biological contents of data objects.

The, at its time, revolutionary idea of providing a common ontology for biological objects along with a central repository for web services descriptions led to the creation of a consistent platform with a broad development support and many tools being developed. Indeed, support of various development languages (Java and Perl) and the availability of development tools provided a universal acceptance by bioinformatics services developers. More than a thousand services covering all sorts of bioinformatics applications can be found in BioMoby Registry along to an extensive Object Ontology which describes hundreds biological datatypes. Surprisingly enough, the availability of development tools that made BioMoby so popular for developers, became a limitation in time of new standards adoption. BioMoby implementation relied on an in-house XML serialization that required a proprietary API (e.g. jMoby). Web Services Definitions (WSDLs) generated by the BioMoby API are not understood by standard programmatic tools and lack the semantic contents that makes BioMoby special. This has restricted the use of BioMoby web services to on-purpose built clients (Gordon & Sensen, 2007) and required the development of specific plug-ins for popular clients like Taverna (Kawas, Senger, & Wilkinson, 2006). The present work has tried to conciliate the BioMoby framework with standard web services technologies, through the development of a new Java API,
and providing a pipeline to adapt the execution of BioMoby services to standard clients, and technologies.
4.1.1. New lightweight Java API for BioMoby Registry access and Web Services execution.

The requirement

Although BioMoby platform was initially based on Perl, Java quickly became to a scene with a jMoby API. The API brought to Java developers an opportunity to create and execute BioMoby web services and provided means to work with BioMoby registry servers.

Even though jMoby API provided all necessary functionality for BioMoby developers, it had its limitations arisen from a custom XML binding framework. Java platform already has a standard XML binding architecture (JAXB) that is tightly integrated with a way how Java-based web services are developed (JAX-WS). Non-standard XML binding leads to incompatibility issues with latest Java Application Servers. Many external dependencies also required non-trivial solutions (Gordon & Sensen, 2007) for BioMoby client applications developers.

Some issues arises from the BioMoby Central SOAP API, which although provides all the functionality to manage BioMoby Registry, does not represent a consistent API where common data structures may be reused by all SOAP operations.

To overcome these restrictions a new lightweight BioMoby API has been developed.
**Implementation**

The implementation consists of several libraries. A core functionality that includes a BioMoby message format parser and web-services execution part enclosed in a *MobyCore*\(^{41}\) library. The part responsible for BioMoby Registry server interactions encapsulated within a *MobyCentral* library. Both libraries are based on Java API for XML-Based Web Services (JAX-WS) that is an integral part of Java 6 platform. The *MobyCore* library is intended to work with pre-generated objects that reflects BioMoby ontology, but also allows a manual BioMoby message creation. In fact it is possible to mix both approaches within a same message construction.

The library implements a BioMoby Asynchronous Services specification based on OASIS Web Services Resource Framework (WSRF). Generated by a *MobyGenerator* utility, Java ontology classes are quite similar to those generated by MoSeS\(^{42}\) tool with a difference that all the XML serialization is done by Java Architecture for XML Binding (JAXB) API.

Because there is no external library dependencies, libraries are very small and provide a very light-weight solution for Java based Rich Applications developments. While it is possible to use the *MobyCore* library for web services development, a BioMoby encoding format (SOAP 1.1 Section 5) is considered obsolete by Web Services Interoperability Organization (WS-I) and has poor support in modern Java servers.

\(^{41}\) [http://sourceforge.net/projects/mobycore/](http://sourceforge.net/projects/mobycore/)
\(^{42}\) [http://search.cpan.org/dist/MOSES-MOBY/](http://search.cpan.org/dist/MOSES-MOBY/)
Features

MobyCore

- RPC-encoded” and “Document-literal” SOAP binding style support.
- Synchronous and asynchronous (through WSRF) BioMoby web services execution.
- May work with or without XML mapped Java datatype classes.

MobyCentral

- Provides all the functionality to work with BioMoby Registry servers.

MobyGenerator

- Generates JAXB based Java annotated BioMoby datatypes classes for usage with MobyCore library.
4.1.2. BioNemus. Creating SAWSDL bioinformatics services based on BioMoby ontology model

Introduction

The popularity of BioMoby platform left an immense heritage in a form of available services. Rewriting these services for W3C Web Services standards compliance would require exceptionable efforts from service providers. BioNemus tool automatically generates WS-I compatible web services using an information provided by BioMoby Registry. Generated web services act as a proxy between clients and original BioMoby ones.

Implementation

BioNemus is implemented as Java 6 applet/application and is based on previously described lightweight BioMoby API. The lightweight BioMoby API uses an XML format for services description that allows java code generation through an XSLT transformation. In fact, the code generation may be upgraded without a need to rebuild the tool, just by a modification of correspondent XSL templates. Generated code is compiled using Java Compiler API (JSR-199)\(^\text{43}\) provided by OpenJDK project\(^\text{44}\) what makes possible BioNemus applet usage within web browsers.

Created by BioNemus Java web services application comply with Java API for XML Web Services (JAX-WS) 2.1 specification\(^\text{45}\). Generated application is packaged as a Java Web Application Archive (WAR) which structure is shown on (Figure 20).


\(^{44}\) [http://openjdk.java.net/groups/compiler/](http://openjdk.java.net/groups/compiler/)

BioMoby datatypes are implemented as Java Architecture for XML Binding (JAXB) 2.1\(^{46}\) annotated Java beans. BioMoby object ontology defines a limited set of basic objects that have their direct mapping into BioNemus datatypes (Table 17).

<table>
<thead>
<tr>
<th>BioMoby object</th>
<th>BioNemus type</th>
<th>XML Schema type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>NemusObject</td>
<td>xs:complexType</td>
</tr>
<tr>
<td>String</td>
<td>NemusString</td>
<td>xs:string</td>
</tr>
<tr>
<td>Integer</td>
<td>NemusInteger</td>
<td>xs:int</td>
</tr>
<tr>
<td>Float</td>
<td>NemusFloat</td>
<td>xs:float</td>
</tr>
<tr>
<td>Boolean</td>
<td>NemusBoolean</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>DateTime</td>
<td>NemusDateTime</td>
<td>xs:dateTime</td>
</tr>
</tbody>
</table>

Table 17. Correspondence between basic BioMoby objects and BioNemus types

Some BioMoby elements are also directly translated in their BioNemus counterparts (Table 18).

<table>
<thead>
<tr>
<th>BioMoby</th>
<th>BioNemus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>nemusId</td>
<td>a biological entity identifier of any kind (i.e., &quot;PDB_ID&quot;, &quot;UNIPROT_ID&quot;)</td>
</tr>
<tr>
<td>namespace</td>
<td>nemusNamespace</td>
<td>a concept of data origin, usually goes along with an identifier (id=&quot;P00807&quot;, namespace=&quot;UniProt&quot;)</td>
</tr>
</tbody>
</table>

SecondaryParameters | parameters | usually specifies additional parameters for a service (ie BLAST parameters in runNCBIBlastp service)
--- | --- | ---
xrefs | reference | A cross reference is an optional component of any object.

Table 18. Correspondence between BioMoby and BioNemus elements

SAWSDL library is implemented as an extension to the JAX-WS Reference Implementation (RI)\(^\text{47}\). OWL 2 serialization library is also based on JAXB 2.1 specification and provides OWL/XML serialization\(^\text{48}\).

BioNemus stores its ontology in a user home directory ($user_home$/BioNemus2Cache/ontology.zip). The ontology.zip file contains a set of XML Schemas with defined ontology datatypes. SAWSDL modelReference attribute is used to unambiguously identify XML Schema elements with their semantic counterparts.

Table 19 XML Schema definition for the MD_Trajectory BioMoby object

\(^{47}\) [http://jax-ws.java.net/](http://jax-ws.java.net/)

\(^{48}\) [http://www.w3.org/TR/owl2-xml-serialization/](http://www.w3.org/TR/owl2-xml-serialization/)
These schemas may be directly used to generate appropriate ontology classes (for example XML Binding Compiler). They also may contain XML annotations.

**Functionality**

The primary purpose of the application is a generation of web services from BioMoby ontology.

BioNemus is a quite sophisticated development tool which provides various options for web services generation:

- Management of XML Schema based datatypes ontology.
- Import of BioMoby datatypes ontology directly from BioMoby repository servers.
- Generation of JAXB based Java classes in accordance with corresponding XML Schema.
- Generation of REST JAVA EE 6 web services based on existent BioMoby web services.
- Generation of SAWSDL / OWL 2 Java EE 5 web services based on existent BioMoby web services.
- Generation of SAWSDL / OWL 2 Java EE 5 web services templates based on ontology XML Schema.

Figure 21. BioNemus functional workflow
- Support of asynchronous BioMoby Web Services through WS-Addressing specification.

While the principal BioNemus feature is the generation of semantically annotated document-literal web services based on BioMoby ontology, it can also generate JAX-WS based web service template providing de-novo web services development capability.

To generate a proxy application it is possible to use command line parameters (Table 20). In this case generation is completely automatic.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-url</td>
<td>URL of the Registry. (if not specified, obtained by -registry namespace)</td>
</tr>
<tr>
<td>-registry</td>
<td>Registry namespace to connect to. (optional, default '<a href="http://www.inab.org/MOBY/Central">http://www.inab.org/MOBY/Central</a>')</td>
</tr>
<tr>
<td>-authority</td>
<td>Authority to generate a proxy. (if not specified, proxy generated for all services)</td>
</tr>
<tr>
<td>-mode</td>
<td>Type of generated proxy. ('METRO', 'SAWSDL', 'REST')</td>
</tr>
<tr>
<td>-fast</td>
<td>Reuse cached datatype information.</td>
</tr>
<tr>
<td>-output</td>
<td>Generated output proxy (*.war) file. (if not specified, $authority + '.war')</td>
</tr>
<tr>
<td>-help</td>
<td>Help about parameters.</td>
</tr>
</tbody>
</table>

Table 20. BioNemus commandline parameters

For instance:

```
>java -jar BioNemus2.jar -mode REST
```

Generates REST-based web services for all INB authorities.

```
>java -jar BioNemus2.jar -authority inb.bsc.es -mode SAWSDL
```

Generates SOAP-based semantically annotated web services for the 'inb.bsc.es' authority.

**Usage of the generated services**

Automatically generated proxy application for BioMoby web services may be deployed into any Java Enterprise Edition compatible Java Application Servers such as JBoss, Glassfish, etc.

There are two different kinds of proxies that can be generated by BioNemus tool:

- Document/Literal SOAP-based Web-services based on JAX-WS 2.1 specification (requires JEE5 compatible server).
- RESTful web services based on JAX-RS 1.0 specification, that demands JEE6 compatible server (i.e. JBoss 6)

For an execution of SAWSDL Document/Literal SOAP-based web services any specification conformal tool may be used. For instance, it is possible to use “wsimport” utility to generate all necessary artifacts:

>wsimport http://www.inab.org/dproxy/inb.bsc.es/runNCBI Blastp?wsdl

The example command generates a “runNCBI Blastp” web service client which includes ontology classes, primitive datatypes and service related artifacts (request, response, fault, etc.).

The generated web service may be executed using standard java JAX-WS API:

```java
NemusString str = new NemusString();
str.setValue("MKELNDLEKKYNAHIGVYALDTKSGKEVKFNSDK");
AminoAcidSequence sequence = new AminoAcidSequence();
sequence.setSequenceString(str);
RunNCBI BlastpRequest request = new RunNCBI BlastpRequest();
request.setSequence(sequence);
RunNCBI Blastp_Service service = new RunNCBI Blastp_Service();
RunNCBI Blastp port = service.getRunNCBI BlastpPort();
RunNCBI BlastpResponse response = port.runNCBI Blastp(request, 1, 180);
String blast = response.getBlastReport().getContent().getValue();
System.out.println(blast);
```

Table 21. Java code example for BLAST web service execution
While it is possible to create a client using only provided WSDL file, generated proxy application already contains necessary artifact libraries:

- /lib/services.jar - web-services interfaces.
- /lib/NemusDatatypes.jar - primitive datatypes
- /lib/NemusOntology.jar - ontology classes

Using these three libraries, it is possible to create a client without the need of “wsimport” utility.

```java
@WebServiceClient
public class RunNCBI_Blastp_Service extends Service {
    public RunNCBI_Blastp_Service(URL wsdl) {
        super(wsdl, new QName("urn:lsid:proxy.bionemus.org:service", "runNCBI_Blastp"));
    }

    @WebEndpoint
    public RunNCBI_Blastp getRunNCBI_BlastpPort() {
        return super.getPort(new QName("urn:lsid:proxy.bionemus.org:service", "runNCBI_BlastpPort"), RunNCBI_Blastp.class);
    }
}
```

Table 22. BLAST web service client

RESTful web services can be also generated by the BioNemus. Based on JAX-RS 1.0 specification they use the same pattern as SOAP ones and are very easy to use. Generated services support both XML Schema and JSON based encoding, thus could be used directly from Javascript. Note that to create an
XML message it is possible to use ontology based java classes along with a JAXB API.

```java
URL url = new URL("http://www.inab.org/dproxy-rest/inb.bsc.es/getEntryFromPDB");
HttpURLConnection conn = (HttpURLConnection)url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/xml");
conn.setRequestProperty("Accept", "application/xml");
conn.setDoOutput(true);
NemusObject id = new NemusObject();
id.setNemusId("1pio");
id.setNemusNamespace(new QName(null, "PDB"));
JAXBContext ctx = JAXBContext.newInstance(NemusObject.class);
OutputStream out = conn.getOutputStream();
ctx.createMarshaller().marshal(id, out);
BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
String line;
while ((line = rd.readLine()) != null) {
    System.out.println(line);
}
```

Table 23. Java getEntryFromPDB RESTful web service execution

```html
<body>
    <script type="text/javascript">
        var url = "http://www.inab.org/dproxy-rest/inb.bsc.es/getEntryFromPDB";
        var http = new XMLHttpRequest();
        http.open("POST", url, true);
        http.setRequestHeader("Content-Type", "application/json");
        http.setRequestHeader("Accept", "application/json");
        http.onreadystatechange = function () {
            if (http.readyState == 4 && http.status == 200) {
                alert(http.responseText);
            }
        }
        var req = '{"nemusId": "1pio", "nemusNamespace": "PDB"}';
        http.send(req);
    </script>
</body>
</html>
```

Table 24. JavaScript getEntryFromPDB RESTful web service execution
Achievements

The main achievement of the project was its practical application for National Institute of Bioinformatics (INB) BioMoby web services collection. Using the BioNemus tool the complete offer of INB BioMoby web services were published as Semantically Annotated Document/Literal ones. In addition RESTful access is also provided with XML and JSON encoding support.

<table>
<thead>
<tr>
<th>Authority</th>
<th>Web Services</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>synchronous</td>
<td>asynchronous</td>
<td>total</td>
</tr>
<tr>
<td><a href="http://www.cnb.csic.es">www.cnb.csic.es</a></td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>inb.bsc.es</td>
<td>110</td>
<td>54</td>
<td>164</td>
</tr>
<tr>
<td>mmb.pcb.ub.es</td>
<td>41</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>cnio.es</td>
<td>6</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>genome.imim.es</td>
<td>21</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>bioinfo.cipf.es</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>pdg.cnb.uam.es</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td><a href="http://www.cnb.uam.es">www.cnb.uam.es</a></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>cgl.imim.es</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>chrimoyo.ac.uma.es</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><a href="http://www.bioinfo.uma.es">www.bioinfo.uma.es</a></td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>total</td>
<td>208</td>
<td>70</td>
<td>278</td>
</tr>
</tbody>
</table>

Table 25. web services by the authority

4.1.3. SAWSDL-based BioMoby ontology integration.

As said above, BioMoby project popularity was largely attributed to its community-driven object ontology. The simplicity of the ontology which provided a minimum of relationships such as inheritance (“is-a”) and composition (“has-a” and “has”) contributed to its quick buildup with hundreds of incorporated objects. Even BioMoby services were SOAP-based, BioMoby didn’t embrace XML Schema as a datatype system, providing a custom message serialization format encapsulated within a SOAP message. This peculiarity required the usage of a special API to gather BioMoby services description. These descriptions may be obtained through BioMoby Registry. Interestingly enough, BioMoby Registry still provides WSDL 1.1 definitions, but since there is no datatype information included, their utility is very limited.

In addition to SOAP-based API to access BioMoby Registry, BioMoby platform provided RDF/OWL based web services descriptions based on myGrid ontology. While XML Schema is the only type system supported by WSDL
specification\textsuperscript{49}, semantic models may be embedded into WSDL descriptors as suggested in SAWSDL recommendation\textsuperscript{50}. To uncover this possibility the tinyMOBY library has been developed providing an elegant way to mix BioMoby WSDL descriptions with OWL/RDF based datatype definitions (Table 26).

\textsuperscript{49} http://www.w3.org/TR/wsdl20-alschemalangs/
\textsuperscript{50} http://www.w3.org/TR/sawsdl/#embedding
Table 26. Embedding MOBY-S datatype definitions in WSDL 2.0 description
**Functionality**

The tinyMOBY library integrates MOBY-S object ontology into BioMoby WSDL 2.0 descriptions using SAWSDL specification. The library is implemented as an extension to tinyWSDL parser and is based on the new lightweight BioMoby Java API. Using the BioMoby API, tinyWSDL may directly connect to various BioMoby repositories for WSDL 2.0 descriptor generation (Table 27).

```java
MobyDescription mobyDescription =
Description description = mobyDescription.getDescription();
```

Table 27. WSDL 2.0 description creation from BioMoby service identifier

Generated WSDL 2.0 descriptor contains the complete information about a correspondent BioMoby service. The tinyMOBY library parses embedded MOBY-S ontology to discover BioMoby service input/output parameters (Table 28).

```java
SAWSDLinterfaceMessageReferenceExtensions ext1 =
    (SAWSDLinterfaceMessageReferenceExtensions)
interfaceOperationInput.getComponentExtensions(WSDLPredefinedExtension.SAWSDL.URI);
SAWSDLElementDeclarationExtensions ext2 =
    ext1.getSAWSDLElementDeclarationExtensions();
List<URI> modelReferences = ext2.getModelReferences();
MobyDescription mobyDescription = new MobyDescription(description);
for (URI modelReference : modelReferences) {
    TypeDefinition type = mobyDescription.getTypeDefinition(modelReference);
    Object param = type.getContent();
}
```

Table 28. Getting BioMoby service input parameters
**Achievements**

The tinyWSDL library has been used in BioSWR project to integrate BioMoby services with more than three hundred services registered. Generated by tinyWSDL, WSDL 2.0 BioMoby service descriptions may be represented in WSDL 2.0 RDF format, thus providing MOBY-S to WSDL 2.0 OWL ontology conversion. The tight integration with the lightweight BioMoby Java API simplifies BioMoby message creation and service execution.
Part II – XML Schema generation from OWL 2 ontologies.

The OWL 2 Web Ontology Language (OWL 2) is quickly gaining popularity as a primary choice for biological ontologies development. Its expressiveness and great tools support offers many advantages over traditionally used in biomedical domain Open Biomedical Ontologies (OBO) format. Many OBO ontologies are moving to OWL 2 providing both versions simultaneously.

Nowadays ontology usage is an ordinary method for biological datatypes classification. Ontologies are extensively used to provide interoperability in Semantic Web Services. Despite the immense interest in RDF/XML format as a type system for Semantic Web Services, XML Schema is the only standard language to define the structure of web services messages. XML Schema provides good data interoperability but suffers from the lack of semantics support. Recognizing the value of semantics, W3C consortium published SAWSDL specification which defines a mechanism for mapping between XML Schema types and semantic data. Proposed for the mapping Extensible Stylesheet Language Transformations (XSLT), present certain difficulties for Schema lowering (semantic model transformation into an XML message) providing that ontologies have very syntactically loose descriptions. The need to maintain both structural and ontological descriptions of biological object definitions requires considerable efforts from Semantic Web Services developers. The OWL2XS tool mitigates this problem, providing an automatic XML Schema generation from OWL 2 ontologies.

4.2.1. Implementation

The OWL2XS tool is implemented in Java language. OWL2XS uses HermiT reasoner for ontology analysis and Apache XML Schema 2.0 library for XML Schema serialization. The tool consists of a Java library and a simple graphical application.

4.2.2. OWL 2 Model to XML Schema transformation

While there are many projects targeted XML Schema to OWL model transformation (Bohring & Auer, 2005) (Tsinaraki & Christodoulakis, 2007), mapping from OWL model to XML Schema is generally considered inconceivable. This disbelief is grounded on inherent difference between two models. The most important obstacle for OWL 2 to XML Schema models

51 http://sourceforge.net/projects/owl2xs/
transformation is in their structural differences. XML Schema is based on tree model while OWL one is a graph based. Semantic Web languages such as OWL are based on open world assumption, where anything that is not explicitly negated is considered as possible. On the other hand XML Schema describes the structure of an XML document and assumes a closed world domain. However, notwithstanding the differences in the models, many similarities may be identified and the transformation is still possible.

Many of OWL 2 entities have corresponding elements in XML Schema and may be directly mapped to their XML Schema counterparts.

**OWL 2 Classes**

Classes are concepts of knowledge domain in which individuals are defined. Classes define categories for instances and may be interpreted as object types. In reference to XML Schema model, OWL classes may be translated into XML Schema complex type elements.

<table>
<thead>
<tr>
<th>OWL 2</th>
<th>XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class data:Sequence</td>
<td><code>&lt;complexType name=&quot;Sequence&quot; sawsdl:modelReference=&quot;http://inb.bsc.es/sobo/data#Sequence&quot;&gt;</code></td>
</tr>
</tbody>
</table>

Table 29. OWL 2 Class representation in XML Schema

**OWL 2 Properties**

Properties are other important components of ontologies. OWL 2 has two main categories of properties – object and data properties. Object properties represent relationships between individuals while Datatype properties relate individuals to data values. Properties are mapped to XML Schema elements. In some cases Datatype properties may also be represented as XML Schema attributes. Because XML Schema attributes may not be substituted, the correspondent Datatype properties may not be a part of properties hierarchy what put serious limitation for future ontology extension.

<table>
<thead>
<tr>
<th>OWL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: data:CleavageSiteAnnotation</td>
</tr>
<tr>
<td>SubClassOf: data:ProteinAnnotation</td>
</tr>
<tr>
<td>and (property:score only xsd:nonNegativeInteger)</td>
</tr>
<tr>
<td>and (property:score max 1 rdfs:Literal)</td>
</tr>
<tr>
<td>and (property:mature_peptide only data:AminoacidSequence)</td>
</tr>
<tr>
<td>and (property:mature_peptide some data:AminoacidSequence)</td>
</tr>
<tr>
<td>and (property:mature_peptide max 1 data:AminoacidSequence)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
### OWL 2 Datatypes

Datatypes are entities that refer to sets of data values. Most OWL 2 datatypes are taken from the set of XML Schema datatypes and thus may be directly used. Custom datatypes are defined as a restriction of built-in ones and may be mapped to XML Schema simpleType element.

<table>
<thead>
<tr>
<th>OWL 2</th>
<th>XML Schema</th>
</tr>
</thead>
</table>
| Datatype format:ClustalW EquivalentTo: (format:MultipleAlignment and xsd:string[pattern "^\s*((\[0-9]+\.[0-9]+\[0-9]+\[0-9]+\[n\r].*)+"^^xsd:string]) | `<complexType name="ClavageSiteAnnotation" >  
  `<extension base="tns:ProteinAnnotation">  
  `<sequence>  
    `<element name="score" type="float"/>  
    `<element name="mature_peptide" type="tns:AminoacidSequence"/>  
  </sequence>  
  `</extension>  
</complexType>` |

Table 30. OWL 2 Properties representation in XML Schema

Table 31. OWL 2 Datatype representation in XML Schema
**OWL 2 Class inheritance**

Inheritance is an important type of class relationships. XML Schema model is tree based and thus does not support multiple inheritance. One of the peculiarities of XML Schema inheritance is presence of two types of inheritance: extension and restriction. The derived type may either “extend” another type (Table 32) by introducing new properties or “restrict” one (Table 33) by putting property constraints. It is impossible to apply both derivation methods simultaneously what may require a creation of an intermediate abstract type (Table 34).

```
Class: data:Sequence
  SubClassOf: data:Data
    and (property:length only xsd:nonNegativeInteger)
    and (property:length max 1 rdfs:Literal)
    and (property:sequence some format:Sequence)
    and (property:sequence only format:Sequence)
    and (property:sequence max 1 format:Sequence)
```

```
<complexType name="Sequence">
  <complexContent>
    <extension base="tns:Data">
      <sequence>
        <element minOccurs="0" name="length" type="nonNegativeInteger"/>
        <element name="sequence" type="ns0:Sequence"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Table 32. XML Schema type extension example
An OWL 2 class with several parents leads to XML Schema complex type with no parents at all (Table 35). All inherited properties are copied into resulted type providing structural equivalence to original OWL 2 class. This approach represents a usual practice in XML Schema development, although it may lead to incorrect schema.
OWL 2

Class: data:MultipleSequenceAlignment
   SubClassOf: data:SequenceAlignment
   and (property:alignment only format:MultipleAlignment)

Class: data:OtherMSA
   SubClassOf: data:MultipleSequenceAlignment
   and (property:alignment only format:OtherMultipleAlignment)
   and (property:metadata only xsd:string)

XML Schema

```xml
<complexType name="MultipleSequenceAlignment">
  <complexContent>
    <restriction base="tns:SequenceAlignment">
      <sequence>
        <element name="alignment" type="ns0:MultipleAlignment"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>

<complexType name="OtherMSA_restriction">
  <complexContent>
    <restriction base="tns:MultipleSequenceAlignment">
      <sequence>
        <element name="alignment" type="ns0:OtherMultipleAlignment"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>

<complexType name="OtherMSA">
  <complexContent>
    <extension base="tns:OtherMSA_restriction">
      <sequence>
        <element name="metadata" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Table 34. XML Schema type inheritance split example
OWL 2

Class: A
SubClassOf: Thing
and (sequence only xsd:string)

Class: B
SubClassOf: Thing
and (annotation only xsd:string)

Class: C
SubClassOf: A, B

XML Schema

```xml
<complexType name="A">
  <sequence>
    <element name="sequence" type="string"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>

<complexType name="B">
  <sequence>
    <element name="annotation" type="string"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>

<complexType name="C">
  <sequence>
    <element name="sequence" type="string"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="annotation" type="string"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

Table 35. XML Schema type inheritance breakage example
4.2.3. Practical applications for bioinformatics Semantic Web Service development

The possibility to develop web services based on properly defined ontology has a special interest from bioinformatics community. To provide bioinformatics developers with a clear path for Semantic Web Services development, a Simple Biological Objects Ontology (SOBO) has been developed.

The ontology follows EDAM architecture, implementing “data”, “format” and “parameter” concepts. It also includes datatype information for proper XML Schema generation.

Generated by the OWL2XS tool XML Schemas reflect SOBO ontology taxonomy and may be immediately used for SWS development. All generated XML Schema types preserve their OWL 2 origins through `sawssdl:modelReference` annotations.

Web services development requires strong programming skills and greatly depends on chosen platform and languages. Java is indeed one of the most popular platforms for web services development. To provide developers with detailed development process example, NCBI blastp web service, based on Java API for XML Web Services (JAX-WS) has been developed52.

JAX-WS web service development usually suggests two approaches, conventionally denoted as “WSDL first” and “java first”. The “WSDL first” approach implies a creation of WSDL service description which is used for

---

52 [http://inb.bsc.es/documents/owl2xs/examples.html](http://inb.bsc.es/documents/owl2xs/examples.html)
automatic java code generation. The disadvantage of the method may be in awkward java code generated for datatypes. JAX-WS delegates the mapping of XML definitions to Java API for XML Bindings (JAXB), which is based on simplified XML Schema model. The “java first” approach is usually used by Java developers when the resulted XML Schema is not supposed to be read by humans. WSDL and related XML Schemas are generated automatically at the time of web service deployment. The disadvantage is that any additional metadata which may be included into web service descriptor is lost.

The approach taken for the example BLAST web service development is mixed. Java representation of XML Schema is generated automatically using XML Java Compiler tool. Then, generated datatypes are used in web service development (“java first”). Finally, the service is instructed to utilize a manually crafted WSDL descriptor with original semantically annotated XML schemas.

Created using the SOBO ontology, BLAST service is a standard SOAP-based document/literal web service, which may be used by any standard tool such as Taverna Workbench or SoapUI, and constitutes a guided example to the creation of web services based on ontology definitions.
Part III – Ontology-based Service Description for bioinformatics integration

Developing ontological specifications for web services description is an important step on the way to Semantic Web Services. The striking number of proposed solutions that have been appeared in the last decade reflects the importance of the subject. Many projects bravely submitted their proposals to W3C where eternalize as submissions. Ontological representation of web services description facilitates service discovery and matching through query languages. On the other hand XML-based description formats are simpler to parse by software agents. WSDL 1.1 is the de-facto standard for web services descriptions and so description ontologies usually provide some degree of affinity. The latter is highly anticipated by service developers since most of development tools are based on it. Rising popularity of RESTful web services puts additional requirements on the ontology to support them.

The diversity of proposed specifications for Semantic Web Services hinders their adoption in Life Sciences, despite the enormous amount of research taking place. Bioinformatics services cataloging and annotation are important challenges already addressed in projects such as the EMBRACE web service collection (Pettifer, et al., 2009) or Biocatalogue (Bhagat, et al., 2010). Providing a standard semantic way to access to the registries may further improve their usability (García, Ruiz, & Cortés, 2012). The experience gained working with several web services registries oriented to life science community allowed to create a clear vision of community needs to be addressed by a modern Semantic Web Registry.
4.3.1. **BioSWR: Semantic Web services Registry for Bioinformatics.**

BioSWR is a new generation web services catalogue based on latest W3C standards. The peculiarity of BioSWR is in its twofold web services representation, traditional WSDL-based and semantical one based on OWL ontology. This distinctive feature reveals Semantic Web potential providing at the same time compatibility with existent web services development tools.

**Implementation**

The Registry is implemented in Java Enterprise Edition 6 platform. JavaServer Faces 2.0 with JBoss RichFaces 4.1 library is used for the Web interface. REST API is implemented using JAX-RS 1.1 specification. SPARQL protocol implementation is based on openRDF Sesame framework (Broekstra, Kampman, & van Harmelen, 2002).

WSDL 1.1 definitions are converted into WSDL 2.0 at the time of registration. Nevertheless, it is still possible to obtain the original semantically enriched WSDL 1.1 definitions via the Registry.

WSDL descriptors may include external WSDLs or XML schemas, which are also stored in the Registry. It should be noted that stored descriptors and schemas are modified to reflect the URLs assigned by the Registry.
Part III – Ontology-based Service Description for bioinformatics integration

The support of RESTful web services is implemented through the WSDL HTTP Binding extension. WADL descriptors are automatically generated for the HTTP-based services.

While descriptions based on provided by the recommendation ontology may be created by any OWL tool, a library that is specially oriented to WSDL to RDF mapping has been created for the project. WSDL2RDF\(^{53}\) library hides OWL/RDF complicity from developers providing an easy and straightforward API for the ontology management. Another advantage of the API usage is in providing a certain level of consistency where introduced elements are verified to be appropriate before incorporation into the ontology. WSDL2RDF library strictly follows the original ontology provided by WSDL 2.0 RDF Mapping specification\(^ {54}\).

**WSDL 2.0 parsing library**

WSDL 2.0, being the most recent W3C specification for web services description has not gained wide acceptance probably because of the little support by software tools. Apache Woden Milestone 9 (Kaputin & Hughes, 2006) and easyWSDL 2.0 (Boissel-Dallier, Lorré, & Benaben, 2009) was investigated to fit the project needs. Woden doesn’t provide an important requirement to manipulate WSDL 2.0 model, while easyWSDL has serious problems with

\(^{53}\) [http://sourceforge.net/projects/wsdl2rdf/](http://sourceforge.net/projects/wsdl2rdf/)

\(^{54}\) [http://www.w3.org/TR/wsdl20-rdf/](http://www.w3.org/TR/wsdl20-rdf/)
extensions parsing. These limitations required a creation of completely new WSDL 2.0 parser library – tinyWSDL55.

The library supports WSDL 2.0 Adjunct extensions (SOAP, HTTP and RPC) and may be integrated with Apache XML Schema library through tinyXMLSchema extension. Apart from the standard WSDL 2.0 extensions tinyWSDL library supports SAWSDL extension. When tinyXMLSchema extension is used it is also possible to provide semantic references for the referenced XML Schema elements.

**Semantic enrichment**

BioSWR provides EDAM Ontology (Ison, et al., 2013) integration through SAWSDL modelReference attributes. The choice of the appropriate annotation subject is defined internally using logical axioms and realized through semantic reasoning (Figure 24).

Apart from SAWSDL references, basic OWL 2 annotation properties such as rdfs:comment, rdfs:seeAlso and rdfs:isDefinedBy are supported. BioSWR keeps track of all annotations, annotating them with rdfs:isDefinedBy (annotation of another annotation). The latter provides flexibility in annotation management, where only authorized authors may modify outdated annotations.

There is no programmatic way to manage semantic annotations, given that standard SPARQL 1.1 Update operations are implemented.

**Semantic data querying and update**

![Figure 24. Example of Semantic Rules definitions](http://sourceforge.net/projects/tinywsdl/)
One of the advantages of providing an ontological representation of web services is the possibility to implement service discovery using query languages. BioSWR provides SPARQL 1.1 protocol implementation for service discovery and annotation (Figure 25).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DESCRIBE ?s WHERE { ?s rdf:type <http://www.w3.org/ns/wswrd-rdf#Service> . }
```

GET (sparql?query=PREFIX%3Drdf%3A%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23%3E-DESCRIBE-%3Fs%3AWHERE%3B%3Fs%3Atype%3A%3Chttp%3A%2F%2Fwswrd-rdf%3AService%3E%3B%37D> HTTP/1.1 200 OK
Content-Type: application/rdf+xml

```xml
<xml version="1.0" encoding="UTF-8">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:wswrd="http://www.w3.org/ns/wswrd-instance#
  xmlns:wsws="http://www.w3.org/2000/01/XMLSchema-instance"
  xmlns:wswr="http://www.w3.org/2001/XMLSchema#">
  <rdf:Description rdf:about="urn:lsid:inb.bsc.es#wsdl.service(runNCBI Blastp)">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">runNCBI Blastp</rdfs:label>
    <wswrd:wsdlLocation rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">.../wswrd实例/service/162A77DF4D559C4EC6260D95172CE85</wswrd:wsdlLocation>
      <wswrd:implementedBy rdf:resource="urn:lsid:inb.bsc.es#wsdl.service(runNCBI Blastp)="/wsdl.Blastp">
    </wswrd:impl>
  </rdf:Description>
</rdf:RDF>
```

Figure 25. SPARQL query example

The query returns a list of all registered web services in RDF/XML format. All results are supplemented with a `wsdl:wsdlLocation` property to locate the original WSDL 2.0 document to localize them in the Registry.

SPARQL 1.1 UPDATE may be used to manage semantic annotations such as `rdfs:comment` and `sawsdl:modelReference` (Figure 26). Note that updates are subject to security restrictions. Only authorized users are allowed to update service annotations. Unless the updated service is marked as “unlocked” only a service owner is allowed to manage its annotations.
Semantic annotations may be removed using a similar procedure (Table 36).

Security credentials may be provided with HTTP request (Table 37).

BioSWR REST API

While SPARQL is used to manage semantic annotations, web services storage is managed by REST-based API (Table 38).
### URL | HTTP Method: | REST method description:
--- | --- | ---
/service/register?url={url} | GET | Registers the WSDL description (either 1.1 or 2.0). WSDL 1.1 definitions are converted into WSDL 2.0 descriptors. Note that while such conversion is not always possible, it should work for most services (SOAP and REST).
/service/register/?lsid={lsid} | GET | Registers BioMoby services by providing its Life Science Identifier. Several BioMoby Registries are consulted to find a service definition.
/service | GET | Gets a complete OWL/RDF ontology containing all registered services.
/service/{id} | GET | Get a web service description by its ID. ID of the registered service is a HEX encoded MD5 hash from the URL/LSID used for the service registration. Note that for BioMoby services, method returns a WSDL as returned from BioMoby Registry. It is possible to retrieve WSDL 2.0, OWL/RDF or WADL description providing HTTP "Accept" header with appropriate MIME type ("application/wsd+xml", "application/rdf+xml" or "application/vnd.sun.wadl+xml").
/service/{id} | DELETE | Deregister the service by its ID.
/service/deregister/{id} | GET | Deregister the service by its ID.

### WADL support

BioSWR supports WSDL HTTP Bindings for RESTful web services descriptions. Given that WSDL HTTP Binding support in web services development tools is close to void, BioSWR provides automatic WADL generation for the services. For simple RESTful web services that can be accessed via internet browser (via HTTP GET request), BioSWR provides a simple URL template that can be easily understood by users. The format of the generated URL is supported by tools like Taverna.

```
<wadl:resources base="http://www.rcsb.org/pdb/rest/">
    <wadl:resource path="describePDB">
        <wadl:method name="GET">
            <wadl:request>
                <wadl:param name="structureId" style="query" type="xs:string"/>
            </wadl:request>
            <wadl:response>
                <wadl:representation
                    xmlns="" element="PDBdescription" mediaType="application/xml"/>
            </wadl:response>
        </wadl:method>
    </wadl:resource>
</wadl:resources>
```
Table 39. WSDL 2.0 and WADL descriptions of the PDBdescription RESTful service
Part IV – Web Services integration into workflows execution tools.

Despite the intrinsic power of web services technology is its integration as external modules in more complex applications, the current bioinformatics use has led less experienced or occasional users to prefer general purpose web service clients. This allows to get almost the same functionality although at the expense of a more manual and less flexible approach. To this end Taverna (Hull, et al., 2006), became the standard for bioinformatics workflow management, either through its interactive interface, or using Taverna Server (Wolstencroft, et al., 2013). More recently, and especially with the dramatic increase of the mount of biological data to be processed, tools based on a personal workbench with data is kept in a single place. Galaxy (Goecks, Nekrutenko, Taylor, & Galaxy, 2010) has become in the last years the election platform for such usage. This section shows the work done in the integration of some of the above technologies in these two platforms.


Taverna Workbench is a popular open source tool for designing and executing workflows. The simplicity of the workflow design and support of SOAP and RESTful web services made it very popular in the bioinformatics community. Given the immense number of available bioinformatics web services, the determination of suitable service may represent a problem for the workflow developer. To improve developers’ productivity Taverna includes the plugin that allows workflow developers to browse services in the BioCatalogue life sciences web services registry from the Workbench and add them to workflows. BioSWR Registry plugin (Figure 27) implements the similar functionality providing in addition the support for the RESTful services and the possibility to annotate them immediately from the Taverna’s Workbench.

56 http://www.taverna.org.uk/documentation/taverna-2-x/taverna-2-x-plugins/#biocatalogue_plugin
57 https://github.com/taverna/taverna-bioswr-perspective
BioSWR Registry plugin is implemented as the Taverna 3.0 OSGi (Open Services Gateway initiative) plug-in and takes advantage of semantic nature of the BioSWR Registry. The Web services list is retrieved as an ontology while annotations are performed via SPARQL update query. The ontology is parsed via the OWL API library. BioSWR plugin uses the tinyWSDL library that has been repackaged as an OSGi component. 58

58 http://moby-dev.inab.org/m2/org/inb/bsc/tiny-wsdl/
4.4.2. Galaxy Gears. Web Services integration into Galaxy workbench.

Modern computational biology analyses are usually comprised of different tasks, and involve many software tools. For more than a decade, Web Services Architecture has been extensively used in Life Sciences as an integration platform to create complex workflows. Many collections, or registries, of bioinformatics Web services have been created to help workflow developers with thousands of available services. The utility of Web services registries cannot be underestimated as they provide the unique point to localize vast amount of computational resources. On the other hand, the popularity of task-based bioinformatics platforms such as Galaxy (Goecks, Nekrutenko, Taylor, & Galaxy, 2010) led to the need of integration of the already available Web services into correspondent workbench environments. The logical step is to use service description information available in existed registries to automate the integration process (Ménager, Kalaš, Rapacki, & Ison, 2015).

With regard to Web services, WSDL descriptors contain complete information for Web services execution, thus making automatic integration very plausible challenge. Galaxy Gears (Figure 28) is a simple graphical application that using provided in WSDL description information automatically generates Galaxy tool definition file.

Figure 28. Galaxy Gears Java graphical tool
The graphical tool analyzes provided WSDL descriptor and displays a simple, flat view of the Web operation parameters. Users are given a possibility to select which parameters must be considered as a workflow data and which should be provided via the interface.

**Implementation**

Galaxy tool is a Java graphical application implemented in Java and based on a custom Apache Taverna `wsdl-generic` library. The library is based on WSDL4\(^59\), Apache Woden\(^60\) and Apache XML Schema 2.0\(^61\) libraries.

**Methods**

Web services execution usually requires the development of a client program for each particular service, what does not suit well for dynamic environments like workflow execution systems. Dynamic Web service execution requires runtime message analysis and construction. The execution engine must thoroughly analyze both, the Web service description, and messages format. To construct Web service message content, all XML elements must be localized. For this purpose XPath references may be used. XPath references allow flat, tabular service parameters representation that fits well for a command line Web service execution utility.

The developed `wsdl-generic` library is a generic library to analyze and execute Web services based on WSDL description file (Figure 29). The library was developed as a part the Apache Taverna project, and distributed under the

![Figure 29. wsdl-generic library architecture](image-url)

---


\(^60\) [http://ws.apache.org/woden](http://ws.apache.org/woden)

Results

As a proof of concept, Basic Local Alignment Search Web service (based on the NCBI BLAST+ tool) was integrated into the INB-BSC Galaxy server. Although Galaxy already has NCBI BLAST+ support (Cock, Chilton, Grüning, Johnson, & Soranzo, 2015), selected Web service has quite sophisticated input parameters structure which makes it a good example for Galaxy Gears. Galaxy Gears advises which properties may be treated as Galaxy data parameters. The automatically generated Web service Galaxy tool (Figure 30) may be incorporated to complex computational pipelines. Galaxy Gears tool is a simple and straightforward way to integrate Web services into Galaxy.

![Figure 30. Generated Galaxy tool interface](http://www.apache.org/licenses/LICENSE-2.0)
“n. A method of confirming others in their errors.”

Ambrose Bierce
Ontology-based data integration is quickly becoming a mainstream approach for biological information sharing and analysis. Ontology languages have become popular for the definition of biological objects, and formal ontologies are nowadays commonly found completing most data management projects in bioinformatics. However, ontology languages were not designed to specify data representation formats. XML Schema is still the only standard to describe Web services messages structure. The need to provide XML messages format along their semantical meaning requires maintaining both structural and ontological definitions for Web services datatypes. This work represents an effort to consolidate the two worlds.

BioMoby ontology has been one of the biggest and most curated bioinformatics ontologies. Along operational descriptions, BioMoby ontology contained a precise data representation syntax, thus providing all necessary information for BioMoby services execution. This has been a unique case where semantic information and data type representation have been combined in a single framework. Other ontology usages largely neglect the strict definition of data types that is required to drive web services usage. Unfortunately, BioMoby did not adopt XML Schema language for its message format description and required special libraries to support the extraction of input data from BioMoby messages, and to construct appropriate messages. This peculiarity impeded standard development tools usage and required additional efforts to learn BioMoby message format. Nevertheless, as said, BioMoby datatype ontology contained all structural information required by XML Schema to formally define biological datatypes in XML format. For this reason, the first, required, step was to develop a new Java library for BioMoby. Developed MobyCore and MobyCentral libraries use JAXB binding framework for the internal representation of BioMoby message and service descriptions and greatly facilitate the development of bioinformatics tools that require BioMoby integration. The libraries have been the basis of further work within the framework made during this thesis. It should be noted that the use of existing standards and libraries, have significantly reduced their footprint (around 200 kb total size). The low footprint and the simplicity of use make them an ideal choice for Rich Internet Applications (RIA) Java development. Following this development, the BioNemus tool overcame BioMoby standards incompatibility problem. Owing to detailed services descriptions provided by BioMoby ontology, it has been possible to generate Web services that fully comply with Web Services Interoperability (WS-I) specification.
BioNemus, developed in a close collaboration with BioMoby development itself, has been designed to serve as a fully automatic interface to BioMoby Web services, making them usable from standard technologies. Additionally, BioNemus includes semantic contents through the use of the SAWSDL technology. Recognizing the popularity of Representational State Transfer (REST) paradigm it also provides a generation of RESTful Web services as an alternative. BioNemus does not have limitations of the original BioMoby platform as most of the internal machinery has been redesigned to be complaint with modern Java standards. This makes BioNemus different from other tools that provide BioMoby integration, and opens the large BioMoby web service existing collection to general bioinformatics developers.

While BioMoby ontology allows naturally for biological datatypes structural definitions, the OWL 2 ontology language itself has enough expressiveness to determine structural information and may be used as a primary language for biological datatypes definition. Moreover, OWL 2 is becoming the preferred format for the definition of new ontologies in bioinformatics. The ability of the OWL 2 to describe datatypes does not eliminate the need in XML Schema. The “ontology first” approach requires a clear understanding of the ontology goal (object datatypes definition) and restrictions this goal may impose. Despite its limitations this method looks very appealing providing that rigorously developed and consistent ontology is a valuable piece of work per se. It should be noted that a lot of previous work has been done in mapping XML Schema to OWL ontology (Anicic, Ivezic, & Marjanovic, 2007). The “schema first” approach certainly has a strong point of being natural for Web Services development, but still requires semantic enrichment of the generated ontologies.

The OWL2XS tool allows an automatic XML Schema generation from properly developed OWL 2 ontologies what greatly speed-up Semantic Web Services development. Some basic rules of the construction of ontologies that targets the XML Schema should be considered. Since the purpose of the ontology is to provide the structural information for the XML document, the multiple inheritance must be strictly avoided. As an illustration of the approach, a simple ontology that defines a set of biological datatypes has been created for the method validation. Using such ontology as a starting point, Web services can be created following a straightforward procedure (see the created example BLAST Web service, section 4.2.3). The project vividly demonstrates an applicability of biological data ontologies for Semantic Web Service development.
While biological objects may be easily described in ontology languages, biological methods or tools usually miss these descriptions. The lack of embedded semantic descriptions makes difficult the building of registries that can be used without human intervention.

Many biological databases lack the ontological representation but provide an access in a form of Web services (Kawashima, Katayama, Sato, & Kanehisa, 2003) (Rose, et al., 2011). In spite of more than a decade of active research, there is still no consensus on standards for Semantic Web Services description. The initiatives come from life science community (BioMoby, mGRID, SADI), industry (OWL-S, WSMO) and W3C Web Services working group (SAWSDL, WSDL 2.0 RDF mapping) and usually do not provide the interoperability. Another serious obstacle has been the lack of quality tools for SWS support. Development of SWS frameworks require many years of software development and testing, so the thorough analysis of available standards and the implementation choices is very important.

Ontology-based Web Services descriptions may provide various benefits over XML-based WSDL language. While WSDL provides details about the format and structure of service messages it lacks semantic information to describe their meaning. Ontology usage may improve Web services discovery and matching by querying over semantic concepts instead of performing a structural service analysis.

On the other hand, WSDL provides precise information about described services and their messages structure. The clear benefit of WSDL 2.0: RDF ontology usage is a possibility to represent Web services in both WSDL 2.0 and OWL/RDF formats.

BioSWR Registry explores this potential providing the ability to work with service descriptors via SPARQL protocol. Unlike iServe (Pedrinaci, Liu, Maleshkova, Lambert, Kopecký, & Domingue, 2010) platform, BioSWR also provides standard WSDL 1.1 / 2.0 descriptors which may be used by common Web services development tools.

The intrinsic consistency between WSDL descriptors and the OWL 2 service ontology is extended with the support of Semantic Annotations for WSDL and XML Schema (SAWSDL) specification. The ability to manage semantic annotations through the SAWSDL specification distinguishes BioSWR from BioCatalogue registry (Bhagat, et al., 2010) which uses keyword tagging. Additionally, to fully complete the adaptation of the BioMoby framework, BioSWR has been provided with a specific BioMoby support. The tinyMOBY library resolves it embedding required datatype information, as obtained from the original BioMoby Registry, into WSDL
descriptions in a form of MOBY-S ontology. Note that embedding semantic models into WSDL was proposed by the SAWSDL specification as a valid extension mechanism. Thus, tinyMOBY library provides an alternative standard way for BioMoby services description.

The final objectives of this work were to provide tools that are available not only for developers, but also for end users. Two main popular platforms have been chosen, Taverna, and Galaxy. Technologies used in the previous developments make Web services, including those generated within the BioMoby standard, usable in any standard Web services client, like Taverna. One of the strong points of Taverna for end users is its ability to interrogate directly web service registries, and therefore, freeing the users of the responsibility of choosing web service providers, and leaving them with the only requirement of building the required workflow. Therefore, the seamless interaction of Taverna with registries is a key feature. BioSWR development has been complemented with a specific OSGI-based plug-in to integrate it into the Taverna workbench. BioSWR WSDL parsing code was incorporated into Taverna’s codebase for the better interoperability. The code was also used in the BioCatalogue project.

The Semantic Web Services Discovery and Provenance approach is not new (Lord, et al., 2004) and was thoroughly researched for more than a decade. The efficiency of the provenance is greatly dependent on the selected semantic model. Although there are several provenance models available such as Open Provenance Model (Moreau, et al., 2011) or the PROV Ontology developed by W3C, Taverna BioSWR plug-in uses the EDAM ontology which is specially targets Life Sciences domain. Nevertheless, BioSWR Registry is based on WSDL 2.0 RDF Mapping ontology with SAWSDL extensions that opens a possibility to use any ontology as a source of semantic descriptions, and hence facilitate web services discovery. The usage of PROV-O ontology may be further considered once the codebase of Apache Taverna 3.0 is stabilized.

Although Galaxy has become a de-facto standard platform in Bioinformatics, it was not designed to cover the use of web services. In fact, Galaxy was mainly designed to deal with large amounts of data, and normally installed as a front-end interface for large data providers. Indeed, the use of distributed web services is largely incompatible with present genomics data, due to data transmission issues. However, publicly available web services offer covers most of the

63 http://www.w3.org/TR/prov-o/
64 http://incubator.apache.org/projects/taverna.html
required functionality. GalaxyGears allows a seamless integration of existing SOAP-based web services into Galaxy workflows. Generated by Galaxy Gears configuration allows to make use of data already contained in the workbench, so web services can be fully combined with traditional Galaxy tools. The expected use of this tool would be the integration of web services, traditionally available at the same data provider’s site that also holds a Galaxy interface.

This doctoral thesis introduces a comprehensive solution for Semantic Web Services development, publishing, annotation and discovery based on the latest W3C standards. The work is a result of long-continued collaboration with many notable SWS projects such as BioMoby, Taverna and the EMBRACE web service collection. Given the amount of technologies integrated, the project also required a cooperation with other projects such as Apache XML Schema, The OWL API, and HermiT reasoner.

The result of the thesis is a creation of Semantic Web Services framework (Figure 31) which involves many developed software libraries, tools and applications that are summarized in the table (Table 40).

Figure 31. Developed frameworks and libraries
<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobyCore &amp; MobyCentral</td>
<td>The lightweight java libraries to execute BioMoby services and to work with BioMoby registries.</td>
</tr>
<tr>
<td>tinyWSDL &amp; tinyXMLSchema</td>
<td>WSDL 2.0 java parser and XML Schema parsing plug-in.</td>
</tr>
<tr>
<td>tinyMOBY</td>
<td>BioMoby WSDL 2.0 integration plug-in for the tinyWSDL parser.</td>
</tr>
<tr>
<td>wsd12rdf</td>
<td>WSDL 2.0: RDF Mapping java library.</td>
</tr>
<tr>
<td>OWL2XS</td>
<td>OWL ontology to XML Schema generation java library.</td>
</tr>
<tr>
<td>BioSWR</td>
<td>Semantic Web Services Registry for Bioinformatics.</td>
</tr>
<tr>
<td>BioNemus</td>
<td>Web Services generation tool based on BioMoby services' descriptions.</td>
</tr>
<tr>
<td>wsdl-generic (experimental)</td>
<td>Experimental version of WSDL 1.1/2.0 Taverna’s library based on XML Schema.</td>
</tr>
<tr>
<td>Taverna BioSWR perspective</td>
<td>The integration of BioSWR into Taverna 3.0 workbench.</td>
</tr>
<tr>
<td>Galaxy Gears</td>
<td>Web Services integration tool for the Galaxy.</td>
</tr>
</tbody>
</table>

Table 40. The complete list of the developed tools

Developed solutions may further improve a quality of web services offered by bioinformatics community.
CONCLUSION

“A conclusion is the place you get to when you’re tired of thinking.”

Jill Shalvis
1. BioNemus project has demonstrated that BioMoby ontology is sufficiently comprehensive to be used as a service descriptions source for automatic WS-I compliant Semantic Web Services generation.

2. OWL2XS project conclusion is that OWL 2 Web Ontology Language provides enough expressibility to thoroughly describe biological objects and can be used as a model for XML Schema definitions.

3. BioSWR project confirmed that Web Services Description Language (WSDL) Version 2.0: RDF Mapping specification is a safe choice for bioinformatics Semantic Web Services Description.

4. tinyMOBY WSDL 2.0 plug-in has shown the benefits of embedding MOBY-S ontology directly into WSDL 2.0 service descriptors, providing indispensable information about the BioMoby message structure, which can’t be described using an XML Schema.
“I quote others only in order the better to express myself.”

Michel de Montaigne


Call for data analysis papers. (2014, Feb). Nature Genetics, 46(3), 213,213.


“Nothing can be loved or hated unless it is first understood.”

Leonardo da Vinci
BioSWR – Semantic Web Services Registry for Bioinformatics

Dmitry Repchevsky\(^1\), Josep Ll. Gelpi\(^{1,2*}\)

\(^1\) Barcelona Supercomputing Center, Life-Sciences Department, National Institute of Bioinformatics, Computational Bioinformatics Node, Barcelona, Spain, \(^2\) Department of Biochemistry and Molecular Biology, University of Barcelona, Barcelona, Spain

**Abstract**

Despite the variety of available Web services registries specially aimed at Life Sciences, their scope is usually restricted to a limited set of well-defined types of services. While dedicated registries are generally tied to a particular format, general-purpose ones are more adherent to standards and usually rely on Web Service Definition Language (WSDL). Although WSDL is quite flexible to support common Web services types, its lack of semantic expressiveness led to various initiatives to describe Web services via ontology languages. Nevertheless, WSDL 2.0 descriptions gained a standard representation based on Web Ontology Language (OWL). BioSWR is a novel Web services registry that provides standard Resource Description Framework (RDF) descriptions and OWL web services descriptions along with the traditional WSDL based ones. The registry provides a Web-based interface for Web services registration, querying and annotation, and is also accessible programmatically via Representational State Transfer (REST) API or using a SPARQL Protocol and RDF Query Language. BioSWR server is located at <http://www.biocas.es/BioSWR/> and its code is available at <https://sourceforge.net/projects/bioswr> under the LGPL license.

**Citation:** Repchevsky D, Gelpi JL (2014) BioSWR – Semantic Web Services Registry for Bioinformatics. PLoS ONE 9(9): e107889. doi:10.1371/journal.pone.0107889

**Editor:** Yu Xue, Huazhong University of Science and Technology, China

**Received:** May 6, 2014; **Accepted:** August 26, 2014; **Published:** September 18, 2014

**Copyright:** © 2014 Repchevsky, Gelpi. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

**Data Availability:** The authors confirm that all data underlying the findings are fully available without restriction. BioSWR Registry is an open source software that is available on SourceForge repository: http://sourceforge.net/projects/bioswr. Other software libraries that are used in the project are also available http://sourceforge.net/projects/wsdltinyrlf/ http://sourceforge.net/projects/tinywSDL. http://sourceforge.net/projects/tinymoloty/, http://sourceforge.net/projects/moelys/

**Funding:** GNE-BSC, National Institute of Bioinformatics, Instituto de Salud Carlos III. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

**Competing Interests:** The authors have declared that no competing interests exist.

* Email: gelpi@ub.edu

**Introduction**

To the extent that the number of Web services available to the Life Science community is continuously growing, there is a need to provide better ways for their description, categorization and discovery. Existing Web services catalogues like EMBRACE [1] or BioCatalogue [2] are usually bound to Web Service Definition Language (WSDL) and limit themselves to annotation of service entities. The need to provide a richer way to describe Web services raised an interest for ontology-based models for Web services description. A large variety of Web services types and protocols either limits the scope of such ontologies to concrete type of services or makes them so abstract that still requires WSDL usage. In the latter case, a link between WSDL components and their semantic descriptions is usually done via Semantic Annotations for WSDL and XML Schema (SAWSDL) annotations [3].

WSDL 2.0 brought a new conceptual model with considerable improvements in Representational State Transfer (REST) Web services description. A possibility to describe RESTful Web services along with Simple Object Access Protocol (SOAP) based ones is a clear step forward especially in life science domain where both approaches are intensively used. Another remarkable improvement of WSDL 2.0 was the introduction of Internationalized Resource Identifiers (IRIs) for its described components. The ability to unambiguously identify every WSDL 2.0 entity as a resource allows using these identifiers within ontologies. WSDL 2.0: RDF Mapping specification provides such ontology to express WSDL 2.0 in Web Ontology Language (OWL). The possibility to describe Web services using standard semantic vocabularies, does not replace the need of ontologies for the life science domain, but rather provides a better integration where Web services may be represented in pure semantic way.

BioSWR registry is a response to the need to introduce a standard semantic view into Web services in addition to the traditional WSDL-based one. OWL/RDF representation of service definitions allows using SPARQL Protocol and RDF Query Language (SPARQL) for Web services discovery and annotation, while WSDL-based representation provides a compatibility with existing Web services development tools.

The choice of WSILD 2.0 as a backbone for Web services descriptions was dictated by the need to support a broader range of Web services. The required easy bidirectional transformation between representational models puts further limitations to the choice of the Web services description ontology. Within several frameworks and specifications aimed at semantic Web services description, W3C Web Services Description Language (WSDL) Version 2.0: RDF Mapping specification has been chosen as a standard for a modern Semantic Web Registry implementation.

**Functionality**

BioSWR is designed to support WSDL 1.1/2.0 Web services, providing a local storage for the registered services and dependent
XML Schema files (Figure 1). While WSDL is generally used to describe SOAP and sometimes RESTful Web services, WSDL 2.0 component model is protocol agnostic and may be adapted to virtually any type of services. To extend a number of supported Web services, BioSWR provides also support for BioMoby [8] services. In the latter case the support is implemented via SAWSDL extension, embedding “Grid BioMoby semantic model [5] into WSDL 2.0 descriptor. BioSWR also relies on SAWSDL for general services annotation, using EMBRACE Data and Methods (EDAM) [6] ontology as the primary source of semantic annotations.

RESTful services are becoming very popular due to their simplicity of use. For this same reason, unlike SOAP-based web-services, they usually lack formal description and are difficult to integrate into workflows. To help developers in the description, and registration of their RESTful web-services, BioSWR documentation provides a simple WSDL 1.1 template. For those RESTful web-services that may be accessed via web browser, BioSWR also provides a sample URL-based template. Finally, for clients requiring more formal descriptions, a WADL description is provided along with the stored WSDL.

BioSWR REST API

BioSWR provides a REST-based API to manage Web services storage (Table 1). The API offers HTTP access to stored Web services definitions that can be directly imported into tools like Taverna [7]. While new Web service registration may be performed by any authenticated user, service removal may be accomplished only by a service owner – the user who originally registered the service. Web service owner may also allow other users to annotate the service, keeping the rights to remove inappropriate annotations. Credentials should be provided via standard basic HTTP Authentication [8].

Semantic data querying

Instead of providing a custom API for Web services search, BioSWR provides SPARQL querying over stored Web services descriptions. BioSWR supports SPARQL 1.1 Protocol query variations via HTTP GET or HTTP POST bindings (Figure 2).

To facilitate SPARQL-based repository discovery, all results are provided with a wssdl:wsdlLocation property to locate the original description document as it was found in the registry. SPARQL query may also be used to filter Web services in the Web interface, however to provide a friendly interface for non-expert users, simple text-based search is available.

SPARQL. UPDATE support provides a simple programmatic way to manage semantic annotations such as rdfs:comment and SAWSDL references (Figure 3).

Semantic enrichment

In accordance with SAWSDL specification, semantic enrichment is attained via sawsdl:modeleReference attributes. The choice of an appropriate annotation subject is defined internally as logical axioms and realized through semantic reasoning (Figure 4). This approach provides flexibility when choosing external annotation sources. BioSWR provides EDAM ontology integration. EDAM was specially designed for bioinformatics/computational biology domain and provides a wide coverage of common bioinformatics objects and methods.

Apart from SAWSDL references, basic OWL 2 annotation properties such as rdfs:comment, rdfs:seeAlso and rdfs:isDefinedBy are supported. BioSWR keeps track of all annotations, annotating them with rdfs:isDefinedBy (annotation of another annotation). The latter provides flexibility in annotation management, where only authorized authors may modify outdated annotations.

BioMoby integration

BioMoby has been a widely used framework to deploy general and specialized bioinformatics WS. Although BioMoby usage has declined over the last years, a large number of services still exist. BioSWR provides BioMoby integration through semantically enriched WSDL 2.0 descriptions. While BioMoby services are SOAP-based they use a special BioMoby message format which cannot be expressed in XML Schema. From SOAP point of view

---

Figure 1. BioSWR general architecture.
doi:10.1371/journal.pone.0107889.g001
its content constitutes an encoded string. Fortunately, BioMoby already provides its own service’s descriptions through BioMoby ""Grid ontology. These definitions are integrated into generated BioMoby WSDL 2.0 descriptors and linked with input/output elements via SAWSDL annotations. The tinyMOBY [10] extension of tinyWSDL [10] parser provides ""Grid semantic annotations management within WSDL 2.0 descriptors. tinyMOBY allows to extract BioMoby data-type definitions from the embedded ""Grid ontology providing a direct integration with BioMoby Java API [11]. The latter eliminates the need of using BioMoby Registries, mostly inactive nowadays, since tinyMOBY datatype definitions includes all required information for BioMoby message preparation and further Web service execution.

WADL support

For WSDL 2.0 services that are described through HTTP Binding Extension, Web Application Description Language (WADL) descriptor may be also obtained via the BioSWR REST API providing an HTTP ""Accept: application/vnd.sun.wadl+xml"" header. The WADL descriptor may be also found in the service description panel of the Web interface.

Web services monitoring

One of the most challenging issues in providing any kind of tool registry in Bioinformatics is to keep track of their availability. There are several levels of Web services monitoring that are usually performed to verify Web services operability. BioSWR implements an availability check inspecting the original Web service description that has been used for the registration. The check is performed periodically or upon user request. The absence of Web service description is interpreted as service withdrawal. Modifications of the original Web service descriptions are detected using cyclic redundancy check (CRC) algorithms [12]. An indication of the status of the service (active, modified, or unavailable) is included in the web interface, and services list can be filtered by such parameter.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DESCRIBE ?s WHERE { ?s rdf:type <http://www.w3.org/ns/wsd1-rdf#Service> . }
```

GET /sparql?query=PREFIX%3Ardf%3A%3Chttp%3A%2F%2Fwww.w3.org%2F%2F2002%2F22-rdf-syntax-ns%2F3E%3EDESCRIIBE%3F%3AWHERE%3B%3Fs%3Adf%3Atype%3B%3Chttp%3A%2F%2Fwww.w3.org%2FsNs%2F%2Fwsdl-rdf%2FService%3E%3B%3D+%3D+

HTTP/1.1 200 OK
Content-Type: application/rdf+xml

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:wssdl="http://www.w3.org/2002/07/owl#"
  xmlns:wadl="http://www.w3.org/ns/wadl-instance#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl-instance#"
  xmlns:xylo="http://www.w3.org/2001/XMLSchemaAnyURI"
  xmlns:xylo:wsidency="http://www.w3.org/2001/XMLSchemaAnyURI"
  xmlns:xylo:wsicore="http://www.w3.org/2001/XMLSchemaAnyURI"
  xmlns:xylo:wssdl="http://www.w3.org/2002/07/owl#">
  <rdf:Description rdf:about="urn:led:imb.bsc.es!/wsdl/service/runNCBI Blast+/">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
    <xylo:wsically:service rdf:resource="urn://192.57.64.19599/EC6624239276C885"/>
  </rdf:Description>
</rdf:RDF>

Figure 2. Find all registered Web services via SPARQL DESCRIBE query.

doi:10.1371/journal.pone.0107889.g002
Implementation

BioSWR is implemented using Java EE 6 Platform. Web interface is based on Java Server Faces 2.0 and RichFaces component framework. BioSWR REST API is implemented using Java API for RESTful Web Services. SPARQL protocol implementation is based on openRDF Sesame framework [15]. Registered services are stored in a MySQL database.

Technologies chosen as a group for the registry became a challenging task of implementation of the latest standards in Semantic Web services. To achieve BioSWR goals several brand-new Java libraries have been developed and contributed to the community:

**wsdl2rdf service description library**

Semantic representation of Web services requires a solid tool to provide a mapping between WSDL 2.0 and OWL model representation. The wsdl2rdf library provides an easy and straightforward API for WSDL 2.0 ontology management, hiding OWL complexity from developers. As WSDL 2.0 ontology does not impose most of the restrictions defined in WSDL 2.0 specification, the advantage of the API usage over a straight ontology manipulation is to provide ontology consistency validation. The wsdl2rdf library strictly follows the original ontology provided by the WSDL 2.0 RDF Mapping specification [14] and is based on the OWL API [15].

**WSDL 2.0 parsing library**

To parse and manipulate WSDL 2.0 descriptions, a brand new WSDL 2.0 library has been developed. The library is based on WSDL 2.0 Part 1: Core Language [16] and WSDL 2.0 Part 2: Adjets specifications [17], and supports both SOAP and HTTP binding extensions. A complementary tinyXmSchema extension library has been developed to provide an easy manipulation of referenced XML Schema elements. The tinyXmSchema library is based on Apache XML Schema 2.0 library [18]. In addition to standard WSDL 2.0 extensions, tinyWSDL supports SAWSDL annotations.

**WSDL 2.0 BioMobity extension library**

In order to provide better integration with BioMobity services, tinyMOBY extension library has been developed. The library allows representing BioMobity services through semantically enriched WSDL 2.0 descriptors. Generated descriptors embed BioMobity RDF definitions that can be used to reconstruct BioMobity message format. As well as the owld2rdf library, tinyMOBY is based on the OWL API. The integration with BioMobity is implemented via lightweight BioMobity Java API.

Figure 3. Add/Remove SAWSDL reference via SPARQL UPDATE query.

doi:10.1371/journal.pone.0107889.g003

Figure 4. Example of Semantic Rules definitions. Here wsdfl Interface saviid #modelReference property is restricted to topic_0003 (Topic).

Because topic_2225 (Protein databases) is a subclass of topic_0003, uritids3id.bsc.es/wwsdl/lsf2ferredgetEntryFromPDB) Interface (which is an individual of wsdfl Interface) may be annotated with it without making the ontology inconsistent.

doi:10.1371/journal.pone.0107889.g004
Discussion

Extending the EMBRACE and BioCatalogue registries philosophies, BioSWR offers many unique features for Semantic Web Services providers and consumers. The most significant advancement of BioSWR is the adoption of WSIDL 2.0 and its standard OWL-based representation. Semantic representation of Web services allows using SPARQL query language for Web services discovery and annotations and greatly simplifies BioSWR REST API eliminating a need in respective methods.

BioMoby was a fairly extended protocol for SOAP-based web services, and a significant number of services are still available. BioSWR offers semantically enriched WSIDL 2.0 descriptors for BioMoby services, freeing BioMoby clients from the need to interact with a BioMoby Central, and simplifying BioMoby services execution.

BioSWR WSIDL 2.0 support also contributes to bringing RESTful Web services back to the standards. Being very popular to access data from repositories due to its simplicity of use, RESTful Web services are often developed outside of the established standards. This precludes those services from being integrated in bioinformatics, and even makes difficult its usage with current clients without a manual adaption. BioSWR has been developed with a special interest in such integration. For RESTful Web services BioSWR provides both WSIDL and WADL descriptions, which can be used by appropriate clients. BioSWR also facilitates URL-based templates for RESTful retrieve operations (HTTP GET verb). As a proof of concept the complete set of RESTful services generated from RCSB have been registered in BioSWR, and an example tutorial using this kind of services in combination with classical SOAP-based is provided.

BioSWR pushes bioinformatics Web Services Registries to a new level of semantic support providing Semantic Web Services descriptions based on their standard OWL/RDF representation. In anticipation of greater SWS adoption by life science community, BioSWR facilitates a smooth transition from conventional WSIDL 1.1 service definitions to OWL-based WSIDL ontology. The use of SAWSDL for semantic annotations provides interoperability with tools that rely on standard WSIDL definitions. The support of SPARQL query language for service discovery, a Web 2.0 single page design, along with a traditional REST-based interface, to register, filter and annotate Web services, makes BioSWR a powerful registry for bioinformatics Web services.

Acknowledgments

The authors would like to thank José María Fernández González for his invaluable help with determination of project requirements and functionality.

Author Contributions

Conceived and designed the experiments: DR JJG. Performed the experiments: DR. Analyzed the data: DR. Contributed to the writing of the manuscript: DR JJG.

References

The EMBRACE web service collection

Steve Pettifer1,2, Jon Ison2, Matúš Kalaš3,4, Dave Thorne5, Philip McDermott1,5, Inge Jonassen3,4, Ali Liaquat9, José M. Fernández6,7, Jose M. Rodriguez6,7, INB-Partners7, David G. Pisano6,7, Christophe Blanchet8, Mahmut Uludag2, Peter Rice2, Edita Bartasevičiute9, Kristoffer Rapacki9, Maarten Hekkelman10, Olivier Sand11, Heinz Stockinger12, Andrew B. Clegg13, Erik Bongcam-Rudloff14, Jean Salzemann15, Vincent Breton15, Teresa K. Attwood1,6, Graham Cameron2 and Gert Vriend10

1School of Computer Science, The University of Manchester, Manchester, M13 9PL, 2EMBL European Bioinformatics Institute, Hinxton, Cambridge, CB10 1SD, UK, 3Computational Biology Unit, Bergen Center for Computational Science, 5008 Bergen, Norway, 4Department of Informatics, University of Bergen, 5008 Bergen, Norway, 5Faculty of Life Sciences, The University of Manchester, Manchester, M13 9PT, UK, 6Spanish National Cancer Research Centre (CNIO), Structural Biology and Bioinformatics Programme, 28029 Madrid, Spain, 7Spanish National Bioinformatics Institute (INB), INB Central Node, 28029 Madrid, Spain, 8Université Lyon 1; CNRS, UMR 5096; IBCP, Institut de Biologie et Chimie des Protéines, Lyon, France, 9Center for Biological Sequence Analysis, Department of Systems Biology, Technical University of Denmark, DK-2800 Lyngby, Denmark, 10OMBI, Radboud University Medical Centre, 26-28 6525 GA, Nijmegen, The Netherlands, 11Service de Bioinformatique des GénoMiques et des Réseaux (BiGRe), Université Libre de Bruxelles, B-1050, Belgium, 12Swiss Institute of Bioinformatics, Vital-IT group, CH-1015 Lausanne, Switzerland, 13Research Department of Structural and Molecular Biology, University College London, London WC1E 6BT, UK, 14The Linnaeus Centre for Bioinformatics Swedish University of Agricultural Sciences, S-750 07 Uppsala, Sweden and 15Clermont Université, Université Blaise Pascal, CNRS/IN2P3, Laboratoire de Physique Corpusculaire, BP10448, F-63000 Clermont-Ferrand, France

Received February 2, 2010; Revised March 29, 2010; Accepted April 7, 2010

ABSTRACT

The EMBRACE (European Model for Bioinformatics Research and Community Education) web service collection is the culmination of a 5-year project that set out to investigate issues involved in developing and deploying web services for use in the life sciences. The project concluded that in order for web services to achieve widespread adoption, standards must be defined for the choice of web service technology, for semantically annotating both service function and the data exchanged, and a mechanism for discovering services must be provided. Building on this, the project developed: EDAM, an ontology for describing life science web services; BioXSD, a schema for exchanging data between services; and a centralized registry (http://www.embraceregistry.net) that collects together around 1000 services developed by the consortium partners. This article presents the current status of the collection and its associated recommendations and standards definitions.

INTRODUCTION

Since the early days of the web, the life science community has embraced its use as a mechanism for sharing data, software and knowledge. The enthusiasm and willingness to exchange both research results and the tools necessary to access, visualize and analyse those data is evident through the ever-growing number of resources reported in the annual web server (1) and database editions (2) of Nucleic Acids Research (NAR). More recently, the need has been recognised to provide not only human-accessible web pages but also programmable access to the same resources via so-called ‘Web services’ (3–6). In terms of experimental scalability and reproducibility, there are obvious advantages in being able to automate access to these remote resources through the use of programming languages or workflow systems, such as Taverna (7) and Kepler (8). However, the ease of web services is not without its problems. Summarizing Hull et al. (7), these have historically included: (i) reliance on complex and evolving underlying technologies prone to generating cryptic error messages, (ii) limited documentation and metadata describing services; (iii) incompatible and inconsistent inputs and outputs between services; and (iv) unpredictable

*To whom correspondence should be addressed. Tel: +44 161 275 6259; Fax: +44 161 275 6294; Email: steve.pettifer@manchester.ac.uk

© The Author(s) 2010. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution-Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.5), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
115

EMBRACE

Table 1. The problems identified in Hull et al. (7) and their corresponding solutions, as developed by the EMBRACE consortium

<table>
<thead>
<tr>
<th>Problem</th>
<th>EMBRACE solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inconsistent use of technology</td>
<td>EMBRACE Technology Recommendation Documents. Provide guidance for producers and consumers on selecting appropriate technologies and standards for developing life science web services.</td>
</tr>
<tr>
<td>Limited service metadata</td>
<td>EDAM Ontology. A vocabulary of terms and relations suitable for annotating the behaviour, inputs and outputs of life science services, and for associating meaning with data exchanged between services.</td>
</tr>
<tr>
<td>Incompatible interfaces</td>
<td>BioXSD data interchange format definition. A mechanism for unifying the exchange of data between bioinformatics services.</td>
</tr>
<tr>
<td>Unreliable services</td>
<td>EMBRACE Active Registry. A mechanism for finding services and for monitoring their performance.</td>
</tr>
</tbody>
</table>

performance/reliability. In an effort to address these problems and to provide the life science community with a collection of coherent and robust bioinformatics web services, in 2005, the European Commission provided funds to establish the EMBRACE (European Model for Bioinformatics Research and Community Education) Network of Excellence. The consortium, consisting of 18 institutions, brought together providers of major tools and databases with experts from the informatics domain. To date, the project has produced almost 1000 services, covering a wide functional spectrum, from traditional programs, such as BLAST (9) and ClustalW (10), through to more domain-specific tools and resources, such as metabolite substructure prediction from GC–MS profiles (11) or the prediction of protein stabilization by introducing prolines into the structure (12).

EMBRACE has developed technologies and recommendations to improve the use and uptake of web services within the life science domain; the relationship of these solutions to the issues identified by Hull et al. (7) are outlined in Table 1.

SELECTING A SUITABLE WEB SERVICE TECHNOLOGY

Establishing a basic form of technological standardization required the consortium to first agree on a consistent definition of ‘Web service’. The original term was defined by the World Wide Web Consortium (W3C; http://www.w3 .org) to describe a specific set of technologies, including: the Extensible Markup Language (XML) to package and serialize data; SOAP (originally an acronym for ‘Simple Object Access Protocol’, but since version 1.2, just a capitalized name) and the HyperText Transfer Protocol (HTTP) to orchestrate communication between client and server; and the Web Service Description Language (WSDL) to describe the programmatic interface of the web service itself (http://www.w3.org/TR/wSDL). Today, the term web service has moved into common usage and its meaning has broadened considerably to encompass numerous other web-based mechanisms and approaches for providing remote programmatic service access. For providers and consumers alike, selecting an appropriate web service technology is a daunting task, requiring an in-depth understanding of numerous complex technological issues and implications. Although, on first inspection, some approaches to web service development appear to provide a relatively straightforward route via which providers may deploy their resources [e.g. REST (13) and traditional XML Remote Procedure Calls (XML-RPC; http://www.xmlrpc.com)], the consortium concluded that, on balance, and in the context of the life sciences, the strict guidelines, industry-supported validation tools, fault-tolerance and explicit description language associated with the original W3C definition provided benefits to the consumer that outweighed any short-term inconvenience to the Web service provider. Furthermore, it was considered that even the variety of options and configurations afforded by the W3C definition was too liberal to be practical; the decision was therefore made to adopt the more tightly specified subset of the W3C specification based on the profile defined by the Web Service Interoperability Organization (WS-I, a standards-defining consortium consisting of many of the major players in the IT industry including IBM, Microsoft, Oracle and Intel; http://www.ws-i.org). In addition, EMBRACE recommended the use of the emerging Semantic Annotations for WSDL (SAWSDL) (14) as a means of more richly describing the behaviour of services. More detailed reasoning behind these decisions is reported by Stockinger et al. (15) and ‘life-science friendly’ advice to the producers and consumers of web services is available in the project’s Technology Recommendation documents, available online at the EMBRACE portal (http://www.embraceregistry.net/standards).

ENABLING SEMANTIC DESCRIPTION OF BIOINFORMATICS SERVICES

EMBRACE Data and Methods (EDAM) is an ontology for bioinformatics tools and data, consisting of a set of defined terms, relationships between these terms, and rules that govern the usage of the terms and their relationships. Terms for the initial version of the ontology were collected from analysis of the following tools and resources:
• the EMBRACE web services;
• SOAP-based services provided by the European Bioinformatics Institute (16);
• the myGrid ontology (17); and
• the NAR database and web server categorizations (2).
An initial version of the ontology is available in Open Biomedical Ontologies (OBO; http://www.obofoundry.org) format and uses the terms, relations and rules defined by the OBO Foundry (18). In its current form, EDAM consists of around 1750 terms, with associated definitions and 14 types of relations. The ontology and associated documentation is available at http://edamontology.sourceforge.net.

Terms in EDAM fall into the top-level categories shown in Table 2.

<table>
<thead>
<tr>
<th>Category</th>
<th>Example terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Sequence analysis; Alignment; Sequencing; Microarrays</td>
</tr>
<tr>
<td>Entity</td>
<td>Gene; Amino acid; Residue cluster; Active site; Atom-atom-interaction</td>
</tr>
<tr>
<td>Tool function</td>
<td>Sequence alignment; Pairwise sequence alignment; Sequence database search</td>
</tr>
<tr>
<td>Data resource</td>
<td>PDB database; GO ontology;</td>
</tr>
<tr>
<td>Data type</td>
<td>Sequence alignment; Sequence record; Comparison matrix; Phylogenetic tree</td>
</tr>
<tr>
<td>Identifier</td>
<td>PDB code; UniProtKB (21) accession number</td>
</tr>
</tbody>
</table>

In the context of web services, EDAM plays two important roles. First, as a source of terms that can be added to WSDL files using the SAWSDL extension attributes, it enables the functions, inputs and outputs of a service to be semantically annotated, leading to improved searching from within repositories and better integration with workflow systems. Second, it enables the detailed data types exchanged by services to be more richly expressed, the benefits of which are described in the following section.

**IMPROVING DATA EXCHANGE BETWEEN SERVICES**

Although human-readable domain-specific text file formats have served a valuable purpose in bioinformatics for many years, the growing interest in interoperable web services has required more ‘computer friendly’ approaches to be developed. XML, which defines a format in which data of arbitrary complexity can be encoded, has become the de facto vehicle for inter-system communication (and indeed, forms the basis for the SOAP, WSDL and SAWSDL protocols mentioned previously). Although data records expressed in XML are typically more verbose and difficult for humans to read than their ‘flat file’ counterparts, numerous significant advantages accrue from its use. Of particular relevance here is the ability to use an XML Schema Document (XSD; http://www.w3.org/XML/Schema) to define the ‘grammar’ required during a particular exchange of data. The adhesion of services to this grammar can automatically be validated by well proven industry standard software libraries, allowing tools to detect garbled or malformed inputs and outputs and preventing errors from propagating through to later stages in a service pipeline.

Detailed, fine-grained description of the exchange format by a dedicated XSD has multiple advantages both for the providers of web services and for their users. Data can be automatically validated without the need for bespoke software, increasing the security and making the service implementation less demanding. Conversion between different formats can be achieved reliably via the Extensible Stylesheet Language Transformation (XSLT; http://www.w3.org/TR/xslt) mechanism. Finally, the fine-grained components of the data types can be semantically annotated by terms from a controlled vocabulary such as EDAM.

Maximum interoperability among the diverse bioinformatics web services located around the world can be achieved by using a common, canonical XML Schema. Defining the standard formats of the main data types, the canonical data model allows users to mix-and-match diverse services freely and without the need to write bespoke programs (sometimes referred to as ‘shims’ (22)) to transform to or from the myriad of legacy data formats. This makes the design of analytical workflows, scripts or programs much simpler, faster and cheaper, reducing the need for specialized personnel with advanced programming skills (Figure 1). On the side of the service providers, the common data model brings ready-made and semantically annotated data types that the developers of new services can, in many cases, use directly.

In a similar vein to EDAM, BioXSD has been developed by analysing the existing web services, tools and various existing data formats, and by consulting the bioinformatics community. Initiated by the EMBRACE partners, BioXSD attempts to serve as the common data model for the most widely used, basic biological data exchanged with web services. The current version covers biological sequences, alignments and sequence annotations with both positional and non-positional features, and in addition, defines formats for references to databases or controlled vocabularies/ontologies, literature citations, bioinformatics accession numbers and identifiers.

The core type definitions are accompanied by a number of generic helper types and recommended names for entities that are yet to be assigned ontological definitions. These everyday bioinformatics data types did not previously have any standard XML representations, despite representing both inputs and outputs of more than two-thirds of the web services developed by the project.

Transformers between the BioXSD and the main community textual or tabular formats are included in the BioXSD development, as well as the compatibility with the OpenBio libraries, such as BioPython and BioPerl.

The rules suggested in BioXSD align with a series of well-known resources, such as the Sequence Ontology, Gene Ontology or the BioSapiens Protein Feature Ontology. Examples of its use to describe GFF3 (http://www.sequenceontology.org/gff3.shtml) and UniProt Knowledgebase features is available at http://www.embraceregistry.net/BioXSD/.
SERVICE MONITORING
In order to give users an indication of service reliability, the project developed an ‘active registry’ capable of monitoring the behaviour of its web service collection, and of notifying consumers and service providers of any problems encountered (23). Providing basic classification and searching mechanisms, the registry has been available since October 2008, and has accumulated around 800 WSDL end point descriptions, representing considerably more than a 1000 bioinformatics ‘services’. As the EMBRACE project draws to a close, the data and functionality of the project’s registry are being transferred to the BioCatalogue system (24) that provides more sophisticated curation, tagging, browsing and searching facilities and offers a sustainable long-term repository for the project’s results.
CONCLUSION

EMBRACE has set the stage for bioinformatics web services. It did so by not only recommending standards and schemas for life science web services, but also by delivering in the region of a 1000 web services that are largely interoperable. Figure 2 shows the relative proportions of web service coverage, broken down into various high-level categories. This substantial collection will provide an incentive for future service providers to adopt the EMBRACE web service recommendations, because doing so makes their services interoperable with a rapidly increasing number of other services. We hope that this step forward in web service technology will allow bioinformaticians all over the world to keep up with the exponentially growing data volumes that the ‘omics revolution’ is producing; and we look forward to future NMR special volumes that hopefully will list many new databases, servers, services and facilities to facilitate research in the life sciences by making use of the web services found in the EMBRACE and BioCatalogue registries.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the contributions of Fred Marcus, Stephan Hohman, Rita Casadio, Matthew Woodwarck, Jacques van Helden and Kay Hofmann for providing constant encouragement and stimulating discussions. European Commission within its FP6 Programme, under the thematic area ‘life sciences, genomics and biotechnology for health’ (contract number LHSG-CT-2004-512092 to EMBRACE project).

FUNDING

This project reached a successful conclusion in no small part thanks to the persistent and patient input from its Project Manager, Kerstin Nyberg. Many scientists contributed by registering web services—their details are available via the project’s registry; we especially acknowledge the input from the EU projects ENFIN and BioSapiens, and the numerous INB partners (http://www.inab.org) for registering very many services. Funding for open access charge: the publication charges will be paid from the EMBRACE budget via an account held at the EBII.

Conflict of interest statement. None declared.

REFERENCES


MoDEL (Molecular Dynamics Extended Library): A Database of Atomistic Molecular Dynamics Trajectories

Tim Meyer,1,2,* Marco O’Abramo,1,6 Adam Hospital,1,3,6 Manuel Rueda,1,7 Carlos Ferrer-Costa,1 Alberto Pérez,1,2 Oliver Carrillo,1 Jordi Camps,1,2,3 Carlos Fenollola,1,3 Dmitry Ričepovskiy,1,4 Josep Lluís Gelí,1,2,3,4 and Modesto Orozco1,2,3,4,7

1Joint BSC-SCF Computational Biology Programme, Institute of Research in Biomedicine, Parc Científic de Barcelona, Baldiri Reixac 10, Barcelona 08028, Spain
2Barcelona Supercomputing Center, Jordi Girona 31, Edificio Torre Girona, Barcelona 08034, Spain
3National Institute of Bioinformatics, Parc Científic de Barcelona, Baldiri Reixac 10, Barcelona 08028, Spain
4Departamento de Bioquímica e Biologias Moleculares, Facultad de Biología, Avda Diagonal 645, Barcelona 08028, Spain
5These authors contributed equally to this work
6Correspondence: modesto@imim.ub.edu
DOI 10.1016/j.cub.2010.07.013

SUMMARY

More than 1700 trajectories of proteins representative of monomeric soluble structures in the protein data bank (PDB) have been obtained by means of state-of-the-art atomistic molecular dynamics simulations in near-physiological conditions. The trajectories and analyses are stored in a large data warehouse, which can be queried for dynamic information on proteins, including interactions. Here, we describe the project and the structure and contents of our database, and provide examples of how it can be used to describe the global flexibility properties of proteins. Basic analyses and trajectories stripped of solvent molecules at a reduced resolution level are available from our web server.

INTRODUCTION

Proteins are large and flexible molecules. Under physiological conditions, they adopt an ensemble of conformations. Flexibility patterns of proteins have been carefully refined by evolution to optimize functionality [Ma and Karplus, 1998; Kuhlman and Baker, 2000; Daniel et al., 2003; Qin et al., 2004; Leo-Macias et al., 2005; Karplus and Kuriyan, 2000; Henzler-Wildman et al., 2007; Goldstein, 2008; Yang et al., 2009]. The similarity of the structural variation found in protein families with that spontaneously sampled during molecular dynamics simulations strongly suggests that protein evolution has used the intrinsic pattern of physical flexibility of proteins when designing new proteins [Leo-Macias et al., 2005; Velazquez-Muriel et al., 2009]. In summary, protein evolution and function is difficult to understand if flexibility is ignored. This explains the intense efforts currently being made to obtain experimental descriptions of protein flexibility. However, despite encouraging advances (Jendritza-Larsen et al., 2010), we are far from achieving a full experimental analysis of protein flexibility, and therefore theoretical approaches are necessary. In this respect, coarse-grained (CG) models coupled to ultrasimplified (pseudo) harmonic potentials have been widely used to obtain rough descriptions of the deformability of proteins [Tirion, 1996; Tozzini, 2006; Bahr and Rader, 2005; Yang et al., 2009; Rueda et al., 2007a; Emperador et al., 2008a]; however, in general, the information derived is of low resolution and tends to overestimate the harmonic nature of equilibrium fluctuations. In principle, more accurate descriptions can be obtained from the use of atomistic molecular dynamics (MD), where atomic-resolution trajectories of proteins are derived from the application of Newton’s equations of motion and physical potential energy functions (McCammon et al., 1977; Brooks et al., 1987). Unfortunately, the practical use of MD has been severely limited by its computational cost and by the problems encountered in the automatic setup of simulations. These limitations would explain why MD is traditionally used to study individual proteins.

During the last half of this decade, the development of new and more efficient simulation engines and the availability of state-of-the-art supercomputer (or GRID) platforms has led several laboratories to add a fourth dimension (time) to structural databases by running atomistic MD simulations on the deposited proteins (or at least in a selected set of highly representative structures). Of the many initiatives started, two have crystallized in extended databases: one in the US: DynamicsDB (Beck et al., 2008; Simms et al., 2008; Kehl et al., 2008; Day et al., 2003) developed by Daggett’s group, and another in Europe: MoDEL (Molecular Dynamics Extended Library), which we present here. These large platforms now offer structural biologists a unique tool to analyze the dynamics of proteins.

OVERVIEW OF THE MODEL PROJECT

The main objective of MoDEL is to provide information on the nanosecond scale dynamics of proteins in near-physiological conditions. This information can then be used for many purposes, ranging from evolutionary studies to biophysical analysis and drug-design processes. In addition, MoDEL is an excellent reference set for calibration, refinement, and validation...
of coarse-grained methods of flexibility (Rueta et al., 2007a; Emperador et al., 2008b) and for the benchmarking of force fields, computer programs, and simulation procedures (Rueta et al., 2007a). MoDEL is an ongoing project whose maintenance and extension is one of the main commitments of our group.

MoDEL (Molecular Dynamics Extended Library) is an acronym that defines a complex infrastructure of software and databases that we have developed over several years (Figure 1). It is divided into the following five main blocks: (1) tools for the automatic setup of MD simulations; (2) tools for validation of trajectories and error detection; (3) data warehouse, comprising a relational database and the underlying trajectories database; (4) tools for basic and advanced analysis; and (5) web server and related web applications. All tools have been built using in-house software combined with external software modules (see Table S1 available online) organized and integrated through a software platform. System preparation, simulation, and analysis modules are also available as web services following the framework of the Spanish National Institute of Bioinformatics (BioMobility, BioMoby Consortium, 2008 [www.imab.org]). The modular nature of the software allows combining all operations in fully automated and highly configurable workflows, thereby minimizing human intervention and facilitating maintenance and update. Also, the web services platform allows the integration with the wide offer of bioinformatics services in the community. Raw data are maintained in their original format in order to maximize compatibility with the software designed by third parties. The MoDEL platform is linked directly to a battery of tools for “in-depth” analysis of trajectories and to our FlexServ platform (http://mmb.pcb.un.es/FlexServ) (Camps et al., 2006), which includes a variety of flexibility analyses from MD ensembles as well as from a variety of Cα representations using either normal modes, Brownian Go-like dynamics or Discrete Molecular Dynamics (dMD) (Rueta et al., 2007a; Emperador et al., 2008b).

Simulations in MoDEL are labeled internally following four criteria: (1) simulated structure; (2) length of the trajectory; (3) force field; and (4) solvent environment. Only cytoplasmic monomeric proteins selected by diversity criteria (see below) are currently available in the database, but extensions of the database to membrane proteins and specific protein families are now under way. At the time of writing this report, the MoDEL database contained more than 1700 protein trajectories, ranging from 10 ns (the shortest) to 1 μs (the longest). The raw trajectories collected represent nearly 18 Tb of data corresponding to around 250,000 residues, 4.5 million protein atoms, and around 19 million water molecules. The computational effort required for the derivation of MoDEL required massive use of the MareNostrum supercomputer at the Barcelona Supercomputing Center (www.bsc.es) and local platforms in our group, and took more than 4 years to reach its current completion state.

TARGET SELECTION

A number of reasonable protocols for the selection of target proteins have been proposed (Day et al., 2003; Ng et al., 2006). Here, we adopted a very simple diversity approach intended to select nonglobular proteins covering the largest possible portion of the PDB. The starting point was the release of the PDB in October 2005 (Bernstein et al., 2000), from which we selected Cluster-9D proteins (i.e., we considered in the following only those proteins with less than 50% sequence identity with other proteins selected for simulation). From this reduced list we then removed the following: (1) all membrane proteins; (2) proteins with gaps in the structure; (3) homo- or homodimeric proteins (on the basis of biological assembly definitions found in PDB, Krissinel and Henrick, 2007); (4) proteins with nonstandard residues (except Ser, Met, and Trp) present in 10% or more of the sequence; (5) proteins containing polymeric or nonconservative ligands difficult to parameterize by automatic procedures (see below). This screening produced a final list of 1595 proteins, which then entered the simulation workflow (see Figure 1). Trajectories that failed standard quality checks (see below) were manually analyzed for potential errors in setup and then either repaired or, if no technical errors were found, labeled as potentially artificial, on the basis of either local or global criteria. A number of replicates for several proteins (typically corresponding to different simulation times or force fields; see below) were obtained, thus yielding a total of 1875 trajectories, which were then submitted to the analysis workflows and stored in the MoDEL database. The proteins selected contained from one to four domains and ranged in size from 19 to 994 residues (a distribution plot of protein sizes is shown as Figure S1). A small subset of MoDEL with 30 representative proteins (Day et al., 2003) was created for benchmarking and exploratory studies (this subset is referred to as μMoDEL, in the rest of the paper). Additional benchmark and validation was done considering five selected proteins: 1cpe, 1kxe, and 1opx as representatives of the three CATH major classes, and two proteins for which very large amount of experimental information on flexibility is available: 1ubq and 2gfh; this ultrasmall set is named nMoDEL in the rest of the paper and was again used for validation purposes. A complete list of proteins (and PDB codes) in the μMoDEL and nMoDEL sets is shown in Table S2.
FORCE-FIELD SELECTION

The selection of the force field is a crucial issue in any MD project and there is no clear indication as to which of the many available force fields is the best for protein analysis. Polarizable force fields are promising tools for a careful description of interactions in the future, but they have not been extensively tested to date and they slow down simulations quite significantly. Thus, researchers use standard nonpolarizable force fields. Force fields are in continuous evolution; however, at the time the project was started the following force fields were the most popular: OPLS-VA (Jorgensen et al., 1990), GROMOS-96 (Hermans et al., 1984; Ott and Meyer, 1999) CHARMM-98 (MacKerell et al., 1995, 1999) and AMBER parm99 (Cornell et al., 1995). Before launching all MoDEL simulations, we evaluated the performance of these four force fields in the iMoDEL subset (Rueda et al., 2007b). The data collected demonstrated that these force fields yield similar trajectories, which provide a good reproduction of the structural and dynamical data experimentally available at that time, including residual dipolar coupling (RDC) and order parameter (S') measures for selected proteins (Rueda et al., 2007b). Additional calculations on the iMoDEL set performed with more recent force fields (parm2003 and parm99fs) confirmed that there is a reasonable consensus between force fields for trajectories started from native structures. This observation suggests that for the time length considered in our project, the considered force fields should provide similar results. Calculations on the entire MoDEL set were then performed using the complementary AMBER parm99 and GAFF force fields, for ease of ligand parameterization. For coherence with parm99 the popular TIP3P model (Jorgensen et al., 1983) was used to represent water molecules. Future results of the MoDEL ensemble will incorporate results obtained with newly developed force fields and local refinements of existing ones. The reader is referred to Rueda et al. (2007b) for a detailed discussion on the performance of MD simulations with different force fields.

SIMULATION SETUP AND TRAJECTORY PRODUCTION

One of the biggest challenges in the project was to define robust, flexible, and automatic procedures for the high-throughput setup of MD simulations. The process should be fast and flexible, mimicking the human-based process of preparing and launching a simulation. The refined setup process is detailed in the Supplemental Experimental Procedures section. It was based on a modular and highly flexible workflow structure that could be easily adapted to user requirements. The pipeline allows the user to launch the simulation at the end of the process, by distinct MD codes (at present time: AMBER [Case et al., 2004], NAMD [Phillips et al., 2005], and GROMACS [Hess et al., 2008]). In addition, an independent web application (MDWeb: A.H., M.O., J.L.G., unpublished data) that includes all functionalities has been developed as a side product of the MoDEL project to help in the automatic (but flexible) setup of MD simulations for nonexpert users.

MD simulations were produced in the isothermal-isobaric ensemble (T = 300K, p = 1 atm). Trajectories for the entire MoDEL solution data set were extended for 10 ns (after equilibration).

The 30 protein iMoDEL data set was extended to 0.1 μs and up to 1 μs for the mMoDEL subset. These long simulations were used for benchmarking purposes and to check the validity of the 10 ns trajectories to represent the local dynamics of proteins around native structures (see below). Additionally, gas phase simulations in the isothermal ensemble (T = 300 K) were performed (0.1 μs long for the iMoDEL subset; and 1 μs long for the mMoDEL subset). Detailed simulation settings are included in the Supplemental Experimental Procedures section.

TRAJECTORY CONTROL

MD simulations are numerical simulations based on a large series of simplifications that can generate nonnegligible uncertainties in the results. Errors are expected to increase as a result of the automatic setup procedure required in high-throughput (HT) production, which implies that careful and critical checking of trajectories is needed. In our experience, the main sources of errors in simulations are related to the following: (1) incorrect decisions during the setup, particularly wrong ionic states, poorly placed solvent, or wrong description of the ligand; (2) errors in the equilibration and heating procedure; (3) technical problems along equilibrated trajectory (problems with SHAKE: extreme velocities, thermal coupling, etc.); and (4) force-field problems. Deviations of trajectories from experimental models might also arise for other reasons, such as local uncertainties in the experimental models, and varying environmental conditions in the simulation and in the experiment (for example: different pH, different ionic strength or protein concentration). Inspection of trajectories allows us to recognize errors derived from technical factors (setup/execution/aliasing/integration/co coupling). However, it is not so easy to determine whether deviation caused by force-field problems and other factors (experimental uncertainties, discrepancies between simulated and experimental conditions, etc.) Thus, our strategy was to scan trajectories for anomalous behavior using simple metrics (see Table S3). This was achieved by inspection of trajectories to identify anomalies caused by technical issues (that can typically be corrected) and those that may arise because of nontechnical reasons. In the first case (35 trajectories in total), simulations were repeated and when the anomalous behavior persisted, they were removed from the database, while in the second approach, simulations were labeled as “anomalous” but were maintained in the database since these trajectories can be of interest to some users, and are relevant, for example, in force-field validation and in the discussion of potential local uncertainties in experimental structural models.

Thus, all trajectories were analyzed for global descriptors (see Supplemental Experimental Procedures and Table S3), such as the absolute and relative rmsd, the TM-score (Zhang and Sali, 2004) the radii of gyration and solvent accessible surface (SAS). They were also analyzed for local descriptors, the number of native contacts, and the secondary structure (see Table S3). Trajectories were analyzed after the first nano-second to check for technical problems in the setup (these usually lead to anomalous diffusion or velocities in protein, ligand, or solvent), which were rare and were easy to correct in most cases. At the end of the simulation, quality analysis was...
repeated and a trajectory was labeled “suspicious” in one of the three categories on the basis of the checklist and thresholds shown in Table S3: (1) potential errors in local structure; (2) potential errors in global structure; and (3) potential errors in both local and global structure. Less than 3% of trajectories in MoDEL display one or several warnings, which the user should not ignore.

**ANALYSIS WORKFLOW**

The mining of 16 Tb of raw data is complex and requires automation of analytical tools and further incorporation of results in a relational database (see below). Two types of calculations can be done on raw trajectories: (1) general/basic analysis, which can be performed without previous knowledge of user requirements; and (2) specialized analysis, which requires user specifications and often the development of specific software. The modular nature of the analysis workflow allows the integration of any kind of analysis (for an exploration of commonly used descriptors, see Supplemental Experimental Procedures). Basic analysis includes information on global and local structure, such as rmsd, TM-score, and radius of gyration, total and partial SASAs, collision cross sections, native contacts, secondary structure, and hydrogen-bond pattern. Dynamic descriptors determined by default include fluctuations in all structural values, B factors, and Lindemann’s index (Zhou et al., 1999, frequencies derived from diagonalization of the mass-weighted covariance matrix), entropies (Schütze, 1993; Andricioaei and Kapustin, 2001; Harris et al., 2001) and all the information derived from principal component analysis (PCA) as described in essential dynamics framework (ED; Amade et al., 1993; Orozco et al., 2009, Noy et al., 2009) (for detailed information, see Supplemental Experimental Procedures). All analyses were done with a battery of in-house codes and external analytical tools (see Table S1), which were organized in modular workflows, thereby allowing the incorporation of additional analytical tools to the pipeline.

Specialized modules for the data mining of trajectories are in constant evolution in the group and currently include routines for the analysis of the following: solvent environment (structure and dynamics of water shells); fitting of MD simulations to mesoscopic models of motion, determining hinge points and correlated motions (Campos et al., 2009); finding caustics and escape channels in protein ensembles based on ensemble Brownian dynamics (Canoria and Orozco, 2008); ensemble docking tools (Gebbi et al., 2001); methods for the prediction of potential protein-protein interaction sites (Fernández-Reicio et al., 2005); and many others.

**STRUCTURE OF THE MoDEL DATA WAREHOUSE AND MANAGEMENT SOFTWARE**

The data management of MoDEL involves the handling of a large number of structures, linkage to publicly available databases, accessing a wide repertoire of analyses for each simulation, and storage of the trajectories in a way that facilitates efficient analysis. Although valid attempts to fully integrate this complex set of data have been reported (Berman et al., 2002, Simmons et al., 2008), the MoDEL data warehouse (see Figure 2A) has been designed using a conservative approach in order to be fully compatible with available software. MoDEL combines the following two approaches: (1) a central relational database and (2) a disk-based raw data repository. The former stores structures, simulation details, analytical results, and references to bioinformatics databases, while the latter stores the trajectories in both AMBER (native trajectory formats for other programs are also supported) and compressed PC2 formats, as well as advanced analytical data. The relational database powers the MoDEL web server, which acts as an interface for access to the analyses. The file-system layout of the repository is designed to maximize the efficiency of data retrieval, exploiting hardware parallelism on access to data when possible.

The relational database comprises four main sections (Figure 2A): structure selection, simulation, fragment selection, and analysis. Structure selection includes data for the simulated systems linked to the necessary sections of the PDB (Berman et al., 2000).
Structure

**MoDEL**: Molecular Dynamics Extended Library

et al., 2003), CATH (Pearl et al., 2005), UniProtKB (The UniProt Consortium, 2010), and through the latter to other available databases (Table S1). Simulation details are stored in the simulation section, which includes references to the software used, force fields and solvent, trajectory parameters, and quality-control data.

Trajectory analyses can be performed with a wide set of criteria, not necessarily known at the time of the design of the database, and storing them efficiently is not trivial. Analysis data are centered in the two last sections: fragment selection and analysis block. The central object for analysis storage (anaysisSet) (see Figure S2) is the combination of simulation, the structure fragment analyzed, and the portion of the trajectory to be analyzed. This scheme allows us to store a wide variety of results from a simple collection of trajectory snapshots to a specific combination of analyses done over several parts of the trajectory or restricted to a specific domain. Again, structure fragments can be defined using a series of database data, like our in-house active sites database (A.H., M.O., J.L.G., unpublished data), domain (PFAM, Finn et al., 2008) or fold (CATH) (Pearl et al., 2005; SCOP) (Murzin et al., 1995) databases, and also functional (Gene Ontology) (The Gene Ontology Consortium, 2000) data (Table S1). Setup and analysis software is adapted to extract that information from the database and perform new simulations and analyses on the basis of the desired criteria (see below). The MoDEL relational database is powered by MySQL 5.1 database manager. A complete Entity relationship schema of the database can be found in Figure S2.

The management software is a fully integrated platform (Figure 2B) with a highly modular core mostly written in PERL, combined with preexisting and third-party software (Table S1). To preserve compatibility with third-party software and eventually to allow for the inclusion of new software packages, data are handled in well-known MD formats (amber native, and NenCDF, http://www.unidata.ucar.edu/software/netcdf/). Modules from the platform have been also wrapped to conform to the BioMoby web services framework. MDMobY, A.H., M.O., J.L.G., unpublished data). The central component of the MoDEL management software is the scheduler (Figure 2B). The scheduler module is fed by a queue of simulations selected on the basis of a variety of criteria. It selects the operation to be performed, calling, in turn, structure setup, simulation, quality control, and analysis modules. The scheduler also takes care of checking the data warehouse to detect unfinished or faulty simulations or analyses and resuming the appropriate operations accordingly. Data from the different modules are handled by a common data manager module. The software platform is modular and multiarchitectural to take advantage of the computational infrastructure available (see Figure S3 for a description of the flow of data and the computer architectures involved). Data among the different hardware platforms are synchronized at the storage level and system calls are done through standard RPC technologies.

**WEB-SERVER STRUCTURE**

The MoDEL web server (http://mmb.pcb.ub.es/MoDEL) (see also Figure S4 for screenshots) is designed to allow access to the MoDEL project from several levels: to raw trajectory data for further in-house analysis, to simulation details, and to previously performed analyses. The server is organized into three sections. The first acts as an entry level and is intended for structure selection. The user can either browse the entire set or search for a specific structure. In addition, the database can be browsed following the CATH fold classification. The search criteria implemented include PDB and UniProt IDs, and keyword searches. It is also possible to search from nonstructural descriptors using a sequence comparison module, based on standard BLAST (Altschul et al., 1990) with settings selected to assure that only highly homologous structures are obtained. Using Blast-based sequence comparison with a limited E-value of 10-1, our website currently provides access to simulations covering around 40% of PDB structures, 8% of UniProtKB sequences, 29% of Human UniProtKB sequences and 30% of DrugBank (Wishart et al., 2006) targets.

Once a structure is selected, the system offers a list of available simulations. Simulations can be downloaded, sent to additional tools either open like FlexServ (Carpo et al., 2009), or restricted like MDWeb (Hospital et al., to be published), MGDRED (Carpio and Orozco, 2008), CMAP (Ielpi et al., 2001), or other programs for further analysis, or instead, data previously analyzed can be retrieved. The web also provides videos and 3D animations of the trajectories for visual analysis and projections on the first five principal components to check the nature of the major deformation movements. All the analysis data (see above) are presented as table values, 3D and 2D plots and 3D data using a Jmol applet (http://www.jmol.org). The MoDEL web server is powered by a .Net application server and is linked to an appropriate database manager and software (see above).

**COMPRESSION AND TRANSFER OF DATA**

The management and transfer of data included in the relational database do not need specific software infrastructure, while the access, storage, management and transfer of raw trajectories are (due the amount of the data) complex problems. The original trajectories with all solvent molecules and atomistic details require storage, but most analyses are done by taking intermediate files created by removing solvent molecules. Dry trajectories are compressed to obtain smaller files that can be transferred with high efficiency through the Internet. The compression is done using our PC4z technology (Meyer et al., 2006), which is based on three main steps: (1) principal component analysis of the original trajectory; (2) determination of the reduced set of eigenvectors explaining a given variance threshold (99% by default in MoDEL); and (3) projection of the original Cartesian coordinates into the essential eigenvector space. PC4z splits the original trajectory into two components: the essential eigenvectors and their projections onto the trajectory. This results in a 5- to 10-fold compression of the Cartesian data since a reduced number of eigenvectors is enough to represent a large percentage of variance (Meyer et al., 2006). Note that the compression procedure does not require the assumption of harmonicity in the trajectory and that the original data can be recovered (with the desired accuracy) by simple back-projection to the Cartesian space (Meyer et al., 2006). MoDEL offers (through its webpage, see above) the possibility to download compressed files (99% variance accuracy for heavy atoms). As described elsewhere (Meyer et al., 2006),
compressed files at 90% accuracy provide results that are, for many purposes, indistinguishable from original trajectories (few tenths of Å in most cases from real structures). The largest deviations appear for proteins displaying conformational changes along the trajectory, where a large percentage of variance is then explained by a single mode. The PCAzip program required for compression/decompression can be downloaded from our website http://mmdc.pcb.uu.nl/software/pcazip, both as source code or precompiled executables.

**RELIABILITY OF MD SIMULATIONS**

A first point of concern in our project was the validation of the MD trajectories deposited in our database. This was done in three stages: (1) convergence in force fields; (2) convergence in simulation time; and (3) similarity between MD results and those derived from the experimental structural model. The first point has been checked in a previous paper (Rueda et al., 2007b), which found that the AMBER-pcm99 force field appears to show sufficient reliability for the time window considered in MoDEL (see discussion above). Concerns on the time convergence of trajectories were addressed by comparing simulations on 10, 100, and 500 ns trajectories for a reduced number of highly representative proteins (see above). The results summarized in Figure 3A demonstrate the good agreement between the structures sampled during 10 and 100 ns trajectories for the MoDEL subset both in local and global terms (the same is found for 500 ns trajectories in rMoDEL). Interestingly, not only structural descriptors but also parameters informative on protein flexibility (such as intermolecular contacts) are very similar in short and long trajectories (Figure 3A). This observation confirms that although 10 ns is too short for full protein relaxation, it is long enough to obtain a reasonable representation of the dynamics of proteins around their equilibrium conformation, even in cases of relatively large proteins (see data for GTase activation protein [GPA], a protein with 447 residues, in Figure 55 and also in Figure 3AI). Finally, given that the typical relaxation times of waters are in the picosecond range (the slowest interchanging waters found have residence times <5 ns), MoDEL simulations should provide a complete sampling of the equilibrium solvent atmosphere around proteins.

Our final concern before accepting the utility of MD simulations was the capacity of trajectories in MoDEL to reproduce the known experimental behavior of proteins. Analysis on a reduced set of proteins (Rueda et al., 2007b) suggested that pmf99 simulations provide reasonable approaches to structural models derived from NMR and X-ray data, to 6 factor profiles, and, when available, to direct NMR dynamic data (see above). The results in Figure 3B, obtained from a large set of proteins, confirm our previous claims and demonstrate that MD simulations accurately reproduce global structural descriptors of proteins, such as the solvent accessible surface area or the radii of gyration. RMSD between simulated and experimental models are in 80% of cases below <3 Å, which is not far from the range of uncertainty expected from the normal structural variation found for proteins in water at room temperature. Furthermore, most deviations between MD ensembles and data obtained from experimental models are located in loops (where greater flexibility and larger uncertainties caused by lattice effects are expected in the experimental models), as noted in the low values of TM scores in our simulations (10% of all contacts for around three-quarters of the database and the small losses of secondary structure (for additional discussion on the quality of MD simulations, see Rueda et al., 2007b). In summary, although caution is always necessary when analyzing MD results, we are quite confident that the MD trajectories stored in the MoDEL database provide a reasonable approximation of the equilibrium conformational ensemble of proteins.

**EXAMPLES OF MODEL DATA MINING FACILITIES**

The MoDEL database allows a powerful analysis of average and time-dependent (in the nanosecond scale) properties of proteins and their solvent environment at various levels of resolution (trace, backbone, heavy atoms, and all atoms) and considering the entire system or parts of it. All the analyses can be crossed with internal data in MoDEL or information in other databases that are linked to it. These features thus allow us, for example, to perform a given analysis restricted to a family in CATH or SCOP, to a given domain in PDB, to structures with some functional annotation in Swissprot or TREFP (http://www.uniprot.org), or to protein families with a specific annotation or specific characteristics in the PDB. As noted above, the MoDEL web server gives access to some general analyses, but the MoDEL data warehouse is accessible for many additional ones, which might require specific input from the user. It is not our purpose here to describe the full proterome dynamics; however, below we give a few examples to illustrate the type of information that can be retrieved from our database.

A detailed analysis of dynamic information on proteins that can be extracted from MoDEL will be described elsewhere.

**Family-Specific Analysis of Protein Dynamics**

The MoDEL relational database allows us to analyze family-dependent structural and flexibility properties, using a wide and flexible definition of the concept “family.” This is efficiently done by querying the database against an external or internal descriptor. For example, the data in Figure 4A show how MoDEL provides information on the relative flexibility (as measured by Lindemann’s index) of equivalent thermodynamic and mesoscopic proteins. Global analysis reveals that thermodynamic proteins display 90% of the global flexibility of mesoscopic protein but that this global change in flexibility is not equally distributed throughout the regions of the protein. Thus, the largest rigidification in thermodynamic compared with mesoscopic proteins is located in the backbone (especially in j helices), while the flexibility of side chains (especially in i helices) is not reduced in the former compared with the latter. Another example of MoDEL data mining is shown in Figure 56, which demonstrate that (1) 40%-60% of the variance in this particular set of proteins can be explained by only five essential deformation movements; (2) no major differences are found in the complexity of the flexibility space when considering distinct CATH families; and (3) large proteins do not necessarily have a more complex flexibility...
**Structure**

MoDEL: Molecular Dynamics Extended Library

(A) Different average descriptors for MD simulations in the MoDEL subset. Blue: 10 ns trajectories, red: 100 ns trajectories, green: experimental data. Content in secondary structure is referred to only.

(B) Comparison of structural parameters obtained from MD simulations and from experimental models (see text for details). We consider no change in the secondary structure when the secondary structure element of the starting structure is still very represented (at least for 0.8 ns) in the last nanosecond of simulation. The Rg(esp) and SASA esp) are calculated using the experimental coordinates as found in the PDB. See also Figure S5, and Table S4.

space that small ones, thereby indicating that variance in large proteins is often organized around a limited number of well-defined massive deformations (for example, large loop oscillations or rotations around hinge points).

**Analysis of the Essential Deformability of Proteins**

The MoDEL database has precomputed the essential dynamics (ED) of proteins, which facilitates the study of protein flexibility by reducing the complexity of the deformability space (Amadei...
MoDEL: Molecular Dynamics Extended Library

Figure 4. Examples of Data Mining in MoDEL

(A) Relative Lindemann’s index between protein heavy atoms in thermophyllic and mesophyllic proteins (see Supplemental Experimental Procedures). To gain extra information, the index is computed for different groups of atoms. The nomenclature X/Y in x-axis refers to X: side chains/hydrogens/all; Y: exposed/buried/all and Z: α-helix/β-sheets/α/α/α. The number of thermophyllic proteins is 30, the remaining proteins present in MoDEL are mesophyllic.

(B) Examples of dynamic domain definition and hinge-point location. Using Laevy’s dynamic method, see http://mmb.pcb.ub.es/FlexServ, for four proteins (each dynamic domain is colored differently). Central plot corresponds to the pathway of correlated movements in a protein perturbed at one random residue (color code ranges from green r = 1 to red r = 0.5; blue means no correlation). The search for correlated motions was done with a width of three residues and a depth of four residues (see Supplemental Experimental Procedure and the FlexServ help http://mmb.pcb.ub.es/FlexServ) for additional details.

(C) Apparent Cα-Cα stiffness constants for four proteins with increasing percentage of β sheet (from left to right-bottom). Significant decay of stiffness constants with increased sequence distance is clear. Indicating the local (in sequence) nature of interdomain contacts. However, the presence of large range effects that lead to important contacts between distant (in sequence) residues is clear. The magnitude or remote interdomain contacts become especially clear in β sheet proteins, where the secondary structure forces H-bond-mediated contacts between distant residues. Some of these remote contacts are marked with arrows in the figure.

(D) Results of using MOGrid and CMRF docking on MoDEL ensembles for three randomly selected diverse proteins: (1MRJ) ribosome-inactivating protein in complex with Adenosine (ADN); (2DRB) super transport protein in complex with Ribose (RIB); and (4THI) Threomucase in complex with 2,3-diaminopyrimidine-4-amine (Pyd). Plots in the first column show channel as red tubes, with the corresponding cavity in orange (only 1 of every 10 1/10 residues computed are displayed for clarity). Second column shows druggability measures performed considering true ligands as proteins. "Drug cavities" are shown in yellow and "Hot spots" (regions accumulating 90% of the population of the drug center of mass) are shown in red. The third column shows CMRF best-scored docking poses (green ligands) with a reference to the known crystal structure (orange ligand), where relevant residues at the binding site are displayed with CPK representation.

See also Figures S6 and S7 and Table S5 for additional examples.

et al., 1993; Orozco et al., 2003; Meyer et al., 2006; Noy et al., 2008). Following the ED formalism, after diagonalization of the MD covariance matrix, a set of eigenvectors and another of eigenvalues are obtained, the first gives information on the nature of essential deformation movements, while the second informs on the variance associated with each of these movements. The eigenvectors/eigenvalues can be manipulated in many ways, from simple visualization to complex comparison metrics. Access to external analysis tools, such as PCAasp (http://mmb.pcb.ub.es/software/pcaasus) or FlexServ (http://mmb.pcb.ub.es/FlexServ) (Camp et al., 2009), allows interesting additional analysis, such as the determination of the degree of anharmonicity in the MD simulation, determined by comparison of ED eigenvectors and those derived from...
diagonalization of a Hessian matrix defined by a simple residue-residue harmonic potential (elastic network model description). It is also possible, for example, to compare the similarity between the deformability pattern of a set of related proteins, or to analyze the similarity between physical deformability (as defined by the MD-derived eigenvectors) and the evolutionary deformability derived from the analysis of the structural changes in protein families (see Viegelahn-Muriel et al., 2009 for discussion). An example of the type of information derived from mining MoDEL with these tools is displayed in Table S5.

**Advanced Analysis of Protein Flexibility**

The MoDEL database is linked with advanced analysis tools implemented in FlexSer (http://mmmb.pcb.ub.es/FlexSer) which allows a complete analysis of protein flexibility. Graphical examples in Figure 4B illustrate how trajectories in MoDEL allow the determination of hinge points, dynamics partition of domains and pathways of concerted motions (see Camps et al., 2009 for details). Several neuroepidemiological descriptors of protein deformability can be derived from these analyses, such as the apparent harmonic force-constants acting on the Cα of proteins with different relative content of α helix and β sheet (see Figure 4C). This type of information can be efficiently used to derive more realistic CG models of protein flexibility, of general or family-specific use (Emperador et al., 2008a; Rueda et al., 2007a; Camps et al., 2009; Emperador et al., 2008b). Many more analyses, like those described here, are possible through an intuitive interface, which provides the user with an accurate definition of the desired type of query or analysis.

**Solvent Analysis**

The MoDEL data warehouse contains structural and dynamic information on the solvent atmosphere around protein, which can also be subject to advanced analysis. For example, we can query our database to determine the number of water molecules in close contact with protein residues, to determine water residence times, diffusion properties, preferred solvation sites, and much more information that can also be determined for any given protein family or group of residues. As an example, Figure S7 summarizes some results obtained from the analysis of the first solvation shell around (j) representative proteins of CATH families 1(e) and 2(b). It was found that all the proteins considered here were well solvated with a typical water density around 0.07 to 0.08 waters/Å^2 (in GADS), which compares with a maximum theoretical density (around 0.1 water/Å^2 for ideally packed waters). Interestingly, our data show that j-proteins have more water molecules in their vicinity than u-proteins, even when the water population is corrected by the solvent accessible surface of the proteins (see Figure S7). This observation demonstrates that there is a quite sizeable amount of water around secondary j sheets, even they are traditionally considered hydrophobic structures. Note that analysis similar to that outlined here can be done considering not the entire bulk of solvent but only distinguished water molecules, for example, those placed in crystal positions or cavities, or those with very slow or fast interchange between first and second solvation shells. In other words, MoDEL allows a complete characterization of the solvent atmosphere around proteins.

**Channel and Cavity Detection**

Advanced analysis tools coupled to MoDEL allow the determination of channels and cavities taking the dynamics of the protein into account. It is therefore possible to detect channels or transient cavities, which are present only on small fractions of the trajectory and, accordingly, might not be detectable in the X-ray structure. The procedure is based on our MDGRID algorithm (Camillero and Orozco, 2008), combined with the use of classical probe particles, which can be as generic as a “soft sphere” or as specific as a full drug. As explained in detail elsewhere (Camillero and Orozco, 2008b), MDGRID takes the snapshots collected along the trajectory, projects them in a common rectangular grid and precomputes the forces that the protein atoms will exert on basic particles (positive charge, negative charge, different van der Waals atoms, etc.) placed at the grid points. These forces are then Boltzmann-averaged and used to determine precomputed accelerations within a Brownian dynamics algorithm. Graphical examples of the type of information derived for a few proteins are provided in Figure 4D (first column). These examples clearly illustrate the power of the technique to trace not only the boundaries of the binding site but also the pathways for interchange of ligand with the environment. Note that since forces are precomputed, MDGRID calculations are extremely fast (multimicrosecond long exploration of channels and cavities in a few minutes in a small desktop/personal computer).

**Drugability and Ligand Docking**

The MDGRID protocol outlined above can be used with small changes to determine the “drugability” of a protein (i.e., the capacity of a protein to bind small molecules with drug-like properties). This type of calculation can be done by taking small drug-like molecules from our local molecular database or alternatively by using known drugs for the targeted proteins. In the first case, the study provides a direct measure of protein drugability, while in the second case it is obtained on the ability of a protein to interact with a family of drug-like compounds. In both cases a secondary product is the definition of major binding sites in target proteins. Information is retrieved considering not static pictures of proteins but dynamic ensembles, which might make accessible cavities which are not visible in a single X-ray structure. Figure 4D (second column) contains a few examples of drugability plots for three randomly selected proteins known to bind small drug-like ligands, and illustrates how the method detects that both will bind ligands and locate the primary binding cavity.

For binding sites of known pharmacological targets the use of docking programs such as ChIMP (Gelbl et al., 2001), can yield potential structures of drug-protein complexes (see some examples in Figure 4D, last column). These are obtained explicitly using the flexibility information on the protein contained in the original MD simulation.

**FINAL REMARKS**

Initiatives such as Dynamoeconomics and MoDEL provide access to molecular dynamics data at the proteome level. Expert and nonexpert users can access trajectories and a variety of analyses that may be difficult to reach by other means, thus saving...
them months of work and computer time. Large MD databases provide a proteome-level view to the molecular physics of proteins, something that is impossible to achieve by other means. Furthermore, the databases and integrated analysis tools can be useful for both the benchmarking of force fields and the development of new CG methods. Last, but not least, the research effort devoted to performing and analyzing MD trajectories in the high-throughput regime has generated an extended software platform that allows straightforward, automatic, and robust access to the technique, and to a variety of analysis tools. Initiatives like that presented here are a step forward in the popularization and rationalization of MD simulations, bringing the technique closer to meeting the new needs of the postgenomic era.

SUPPLEMENTAL INFORMATION

Supplemental information includes Experimental Procedures, eight figures, and four tables and can be found online at 10.1016/j.2010-07-013.

ACKNOWLEDGMENTS

MoDEL is a massive effort involving, directly or indirectly, a large part of the Molecular Modeling and Bioinformatics group at IRB Barcelona and the BSC. We are also indebted to Dr. Sergio Gómez and the Manuscript support team for making this project possible. Helpful comments from Prof. F. J. Luque and many colleagues at IRB Barcelona and the BSC are gratefully acknowledged. This work was supported by the Spanish Ministry of Science (CTQ2005-09361-C02-02, BQU2006-10646) Ribi-Sanismo España, the Consolider-Escience project, EU Scalable project, the COMBINED RETICS project and the Fundación Marcelino Botín.

Received: January 23, 2010
Rexivied: July 11, 2010
Accepted: July 27, 2010
Published: November 9, 2010

REFERENCES


Structure

MoDeL: Molecular Dynamics Extended Library


BioNemus: Creating SAWSDL bioinformatics services based on BioMoby ontology model

Dmitry Repchevsky¹, Josep LL Gelpi¹,²,³

¹Barcelona Supercomputing Center, Life-Sciences Department, National Institute of Bioinformatics, Computational Bioinformatics Node, Nexus II Jordi Girona 29, 08034 Barcelona, Spain.
²Dept. Biochemistry and Molecular Biology, University of Barcelona. Av. Diagonal 645, Barcelona 08028, Spain.
³Corresponding author

E-mail addresses:
DR: dmitry.repchevsky@bsc.es
JL: gelpi@bsc.es

Abstract

Background: Web services have been generally accepted for programmatic access to bioinformatics tools and data repositories, especially when large amounts of data should be handled. From the available web services protocols, the increasingly popular RESTful web services may pose a big challenge to those service providers that made significant investments in web services platforms in the early stages of web services development. The present work provides a transition path from one of legacy platforms, BioMoby, towards modern standard web services technologies. One of the most relevant characteristics of BioMoby, its rich semantic foundations, is captured here through the use of the SAWSDL protocol.

Results: The BioNemus tool performs an automatic web services creation wrapping already existent BioMoby services. It has been successfully used for the translation of INB’s BioMoby services collection. The ability to create traditional SOAP-based web services along with RESTful ones provides an additional value for clients. Conclusion: The tool provides an easy way to create modern bioinformatics web services. It is especially useful for those web services providers that have done a significant investment in BioMoby services. It may be also useful for de novo RESTful and SOAP-based web services development based on existent BioMoby datatype system.

Keywords
BioMoby, Web Services, REST, SOAP, SAWSDL, OWL, Semantics.
Background

BioMoby [1] platform was indeed one of the most remarkable projects in bioinformatics web services development. Availability of good tools, support of such popular languages as Java and Perl and an easy development process made BioMoby the chosen platform for many bioinformatics institutions. For instance, “BioMoby” is still one of the most abundant tags in the BioCatalogue web services collection [2].

BioMoby offered an open, ontology-based approach for web services development, where bioinformatics community participated in the ontology evolution via integration of new biological datatypes. To manage the ontology, BioMoby service providers could use a graphical “BioMoby Dashboard” application [3]. BioMoby Central registry also provided a simple SOAP-based API which implemented an essential functionality for the ontology management, annotation and querying.

Although BioMoby web services relied on Simple Object Access Protocol (SOAP, [4]) protocol, they did not employ a XML Schema [5]. Instead, it used a proprietary XML-based serialization format. The custom serialization required a BioMoby API and restricted BioMoby web services usage to on-purpose built clients like Seahawk [6], MOWServ [7], jORCA [8], although more general tools like Taverna incorporated the appropriate plug-ins [9]. The simplicity of BioMoby web services development made it a basement of many bioinformatics platforms [10, 11, 12].

Web services standardization process driven by Web Services Interoperability Organization (WS-I [13]) and increasing popularity of RESTful approach have risen concerns about interoperability of BioMoby web services [14], and contributed to the progressive decay of BioMoby usage. Rewriting BioMoby web services for WS-I compliance would require exceptional efforts from service providers and could be impractical without a proper software support. The Spanish National Bioinformatics Institute contributed with over 250 services to the platform and it is still maintaining the central BioMoby catalogue [15]. Migration of BioMoby web services towards WSDL-based ones requires the adoption of a standard XML Schema to hold BioMoby datatype definitions. Conveniently, XML Schema has all the functionality required to describe BioMoby object ontology. BioMoby object ontology defined only a limited set of relationships (“IS_A”, “HAS_A” and “HAS”) and its primitive types are already restricted to those defined in XML Schema (Error! Reference source not found.) that makes the conversion straightforward.

<table>
<thead>
<tr>
<th>BioMoby datatypes</th>
<th>BioNemas simple types</th>
<th>XML Schema base types</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>NemasString</td>
<td>xs:string</td>
</tr>
<tr>
<td>Integer</td>
<td>NemasInteger</td>
<td>xs:int</td>
</tr>
<tr>
<td>Float</td>
<td>NemasFloat</td>
<td>xs:float</td>
</tr>
<tr>
<td>Boolean</td>
<td>NemasBoolean</td>
<td>xs:boolean</td>
</tr>
<tr>
<td>Datetime</td>
<td>NemasDateTime</td>
<td>xs:dateTime</td>
</tr>
</tbody>
</table>

BioNemas tool automates web services generation process using BioMoby Registries as a source of services descriptions. Generated web services work as a
proxy to original BioMoby services and may be directly consumed by tools like Taverna [16] with no need of a special plug-in usage. Additionally, BioNemus provides a web-services generation framework that can be used to create new WS from scratch, or from the BioMoby data-types ontology.

**Implementation**

BioNemus is implemented as Java applet/application (figure 1). The interaction with BioMoby registries is implemented via the MobyCore library [17].

![Figure 1. Screenshot of BioNemus GUI.](image1)

The right part of the GUI is responsible for BioMoby ontology processing (web services/datatypes generation). The left one is the BioNemus XML Schema based datatype editor, which is used for de-novo web services generation.

Web services Generated by BioNemus target Java Platform, Enterprise Edition 5 (figure 2) and should work on any compatible application server.

![Figure 2. Scheme of BioNemus functional blocks.](image2)

To automatically generate Web services, BioNemus analyses the corresponding BioMoby descriptions, obtained from the BioMoby Registry and generates XML Schema based datatypes. Based on these datatypes, it generates either RESTful or SOAP based Web services that are deployed to the Java Application Server.
Code generation

All code generation is performed via XSL Transformation [18] templates (figure 3).

<table>
<thead>
<tr>
<th>BioMoby AminoAcidSequence datatype XML definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;object_type name=&quot;AminoAcidSequence&quot; xmlns=&quot;urn:biotools.illumina.com&quot;</code></td>
</tr>
<tr>
<td><code>contact_email=&quot;mark@illumina.com&quot; description=&quot;Lightweight representation an amino acid sequence&quot;</code></td>
</tr>
<tr>
<td><code>id=&quot;urn:biomoby.org:objectclass:AminoAcidSequence:2001-09-21T16:00:00Z&quot;</code></td>
</tr>
<tr>
<td><code>&lt;relationship&gt;</code></td>
</tr>
<tr>
<td><code>@object_type article_name=&quot;GenericSequence&quot;</code></td>
</tr>
<tr>
<td><code>@id=&quot;urn:biomoby.org:objectclass:GenericSequence:2001-09-21T16:00:00Z&quot;</code></td>
</tr>
<tr>
<td><code>@relationship_type=&quot;BFA&quot; @relationship_type=&quot;BFA&quot;</code></td>
</tr>
<tr>
<td><code>&lt;/relationship&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/object_type&gt;</code></td>
</tr>
</tbody>
</table>

Applied XSL transformation

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  ...
</xsl:stylesheet>
```

Generated AminoAcidSequence Java class

```
package org.biomoby.objectclass;

import java.util.List;

public class AminoAcidSequence extends GenericSequence implements Serializable {
  ...
```

Figure 3. Java code generation example.

An example of java class generation via XSL transformation of the BioMoby “AminoAcidSequence” XML datatype definition.

Instead of javac compiler, BioNemus uses Java™ Compiler API (JSR-199) provided by the OpenJDK project. This approach allows deploying BioNemus tool as a java applet.

WS-I compliant SOAP-based Web Services

BioNemus produces JAX-WS 2.1 [19] SOAP-based web services that can be deployed on any Java EE 5 compliant Application Server.

BioMoby web services may support asynchronous execution through the Web Services Resource Framework (WSRF) specification. BioNemus generated SOAP services may also support the asynchrnity via Web Services Addressing (WS-Addressing) specification.
When generated for JAX-WS Reference Implementation (METRO), web services may preserve BioMoby annotations using Semantic Annotations for WSDL (SAWSDL) specification [20]. SAWSDL implementation is developed as a JAX-WS RI extension package and provides semantic descriptions for WSDL 1.1 elements. The library automatically generates OWL 2 annotations embedding them into the generated WSDL [21] file.

OWL2 XML Serialization [22] is provided through a JAXB [23] based Java library.

**RESTful Web Services**

BioNemus generates JAX-RS 1.0 [24] based RESTful web services that can be deployed on any Java EE 6 compliant Application Server. Despite the differences between JAX-WS and JAX-RS programming models, BioNemus seeks to keep similar interfaces for both SOAP and RESTful web services. Although both types of services use the same XML datatypes, RESTful web services may consume and produce JavaScript Object Notation (JSON) objects. Natively understood by JavaScript, JSON is nowadays widely used in Asynchronous JavaScript requests (AJAX) and provides a convenient way to consume RESTful web services directly from HTML browsers (see figure 4 for an example call to a RESTful WS using the Ajax request).

```html
<script type="text/javascript">
     var url = "http://www.jnab.org/dproxy/rest/job_dwc.es/getEntryFromPDB";
     var http = new XMLHttpRequest();
     http.open("GET", url, false);
     http.setRequestHeader("Content-Type", "application/json");
     http.setRequestHeader("Accept", "text/plain");
     http.onreadystatechange = function () {
         if (http.readyState === 4 && http.status === 200) {
             var res = JSON.parse(http.responseText);
             alert(res.content.value);
         }
     }
     var req = "";
     http.send(req);
</script>
</html>
```

Figure 4. getEntryFromPDB RESTful Web service execution.

This is a short example which executes an automatically generated getEntryFromPDB RESTful Web service via an AJAX JavaScript request. The script just shows “1P1O” PDB into the popup message.

**XML Schema generation and management**

BioNemus creates a set of JAXB annotated Java bean classes that reflect BioMoby object ontology. SOAP and RESTful web services rely on JAXB for the data binding and mappings between Java objects and XML documents. In a similar way, BioNemus uses JAXB to create an XML Schema that represents BioMoby datatype ontology. This XML schema may be modified for the purpose of manual web service creation.

BioNemus stores its XML schemas locally in the user’s home directory ($USER_HOME$/.BioNemus2Cache/ontology.zip). The ontology.zip file contains a set of XML schemas with defined ontology datatypes (see figure 5 for an example of
XML schema that corresponds to the AminoAcidSequence BioMoby data type object.

Figure 5. AminoAcidSequence XML Schema element

An example of automatically generated AminoAcidSequence XML Schema element, based on the information extracted from BioMoby datatypes ontology. Semantical annotations are preserved as XML Schema ones.

**Functionality**

The principal goal of the BioNemus tool is to provide an easy way for web services transition from the BioMoby platform towards W3C standard-based web services solutions. The application delivers an extensive set of functionality for web services generation:

- Management of XML Schema based datatypes ontology.
- Import of BioMoby datatypes ontology directly from BioMoby repository servers.
- Generation of JAXB based Java classes in accordance with corresponding XML Schema.
- Generation of RESTful JAVA EE 6 web services (automatic for existing BioMoby web services).
- Generation of SAWSDL / OWL 2 Java EE 5 web services (automatic for existing BioMoby web services).
- Generation of SAWSDL / OWL 2 Java EE 5 web services templates based on the generated ontology XML Schema.
- Support of asynchronous BioMoby web services through WS-Addressing specification.

In addition to the graphical interface BioNemus provides also a command line one [25].
Results
Extending a life-cycle of the mature BioMoby platform via providing multiple interfaces for already existent services is an inexpensive way to fulfill interoperability requirements as well as to satisfy a growing interest in RESTful web services.

Practical application
BioNemus has been successfully used by Spanish National Institute of Bioinformatics (INB), providing RESTful and SOAP document/literal interfaces to its large BioMoby web service collection (Error! Reference source not found.). Web services automatically generated by BioNemus also include a description page with brief services descriptions [26].

Table 2 The number of Web Services by their authority

<table>
<thead>
<tr>
<th>Authority</th>
<th>Web Services</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>synchronous</td>
<td>asynchronous</td>
<td>total</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.enb.csic.es">www.enb.csic.es</a></td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>inb.bsc.es</td>
<td>110</td>
<td>54</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>mnb.pcb.ub.es</td>
<td>41</td>
<td>1</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>cnio.es</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>genome.imim.es</td>
<td>21</td>
<td>0</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>bionfo.cifp.es</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>pdc.cnb.uma.es</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.cnb.uma.es">www.cnb.uma.es</a></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>cgl.imim.es</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>chrnayo.ac.uma.es</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.bionfo.uma.es">www.bionfo.uma.es</a></td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>208</td>
<td>70</td>
<td>278</td>
<td></td>
</tr>
</tbody>
</table>

Web services provided by the “inb.bsc.es” authority have been registered in public web services catalogue for the life sciences [2, 27] extending services dissemination and visibility.

Workflows participation
One of the most valuable aspects of BioMoby was the ontology of biological objects that provided common datatypes for BioMoby services. Shared datatypes allows designing complex web services interactions in popular tools like Taverna workbench. SOAP-based web services, as generated by BioNemus, are WSDL compliant and may be included in Taverna workflows directly (figure 6), thus making the original (and now outdated) BioMoby plugin nonessential.
Discussion

BioMoby platform was one of the most ambitious projects that provided an open framework for bioinformatics web services development, publishing, discovery and integration. The platform benefited from a wide acceptance and had plenty of services developed by community members. Despite the wide support in the past, the particularity of the platform required a significant effort to maintain. Although an official BioMoby repository is still available [15], its usage is in a clear decay, in favor of more usable RESTful services. This requires however to recode most of the applications to adopt the new interface. The proposed solution is a simple and straightforward way to reutilize the enormous amount of work invested in BioMoby web services development.

Although web services generation tools for life sciences are not new and has been used by bioinformatics service providers, BioNemus is a unique tool that takes an advantage of BioMoby ontology for automatic web services generation.
Conclusions

One of the strongest points and the major achievement of the project is a seamless integration with BioMoby platform and a possibility to reuse its ontology and already existent services. BioNemus provides a solid platform for bioinformatics web service developers bringing a development to a new level of interoperability. The adoption of BioNemus at the Spanish National Institute of Bioinformatics has greatly simplified the web services access and significantly enhanced web services viability.

Availability and requirements

Project name: BioNemus

Project home page: http://irb.bsc.es/documents/bionemus2/

Operating system(s): Platform independent

Programming language: Java

Other requirements: Java 6.04+ or higher

License: LGPL

Any restrictions to use by non-academics: None

List of abbreviations


Competing interests

The authors declare that they have no competing interests.

Authors' contributions

DR developed BioNemus and wrote an initial draft of manuscript. JL provided project supervision, architecture decisions and revised the manuscript. All authors read and approved the final manuscript.

Acknowledgements

The authors would like to thank Romina Royo for rigorous user feedback and testing as well as José María Fernández for INB services deployment and administration.
References

19. JSR 224: Java\textsuperscript{TM} API for XML-Based Web Services (JAX-WS) 2.0
20. Semantic Annotations for WSDL and XML Schema
   \url{http://www.w3.org/TR/sawSDL}, Accessed 3\textsuperscript{rd} Aug 2015
21. Web Services Description Language (WSDL) 1.1 \url{http://www.w3.org/TR/wSDL},
    Accessed 3\textsuperscript{rd} Aug 2015
22. OWL 2 Web Ontology Language XML Serialization (Second Edition)
   \url{http://www.w3.org/TR/owl2-xml-serialization}, Accessed 3\textsuperscript{rd} Aug 2015
23. JSR 222: Java\textsuperscript{TM} Architecture for XML Binding (JAXB) 2.0
24. JSR 311: JAX-RS: The Java\textsuperscript{TM} API for RESTful Web Services
    Accessed 3\textsuperscript{rd} Aug 2015
26. INB-BSC Web Services \url{http://www.inrb.org/dproxy/inrb.bsc.es/}, Accessed 3\textsuperscript{rd}
    Aug 2015
27. Repchevsky D, Gelpi JL. BioSWR – Semantic Web services Registry for
OWL2XS: Generation of XML Schema from OWL 2 Web Ontology Language.

D.Repchevsky1, Jon Ison2, J L Gelp1*1,3
1Barcelona Supercomputing Center, Life-Sciences Department, National Institute of Bioinformatics, Computational Bioinformatics Node, Barcelona, Spain, 2EMBL European Bioinformatics Institute, Hinxton, CB10 1SD, UK, 3Dept. Biochemistry and Molecular Biology, University of Barcelona.

Summary: Web services are a popular architecture for bioinformatics software development. Ontologies, represented using a description language such as Open Biomedical Ontologies (OBO) or Web Ontology Language (OWL), may be used as a source of annotations for web services through the Semantic Annotations for WSDL (SAWSDL) mechanism, while XML Schema is still used for defining the message structure. The OWL2XS tool automatically generates a semantically annotated XML Schema from an OWL 2 ontology, greatly facilitating semantic web service development where an ontology but no formal XML schema is available.

Availability and Implementation: OWL2XS consists of a Java library, and a graphical interface implemented as a Java Applet/Application. The tool is freely available at:
http://nb.bei.ece.documents/ow2xs/index.html
*To whom correspondence should be addressed

INTRODUCTION

Despite the increasing interest in emerging data formats like JSON and YAML, XML is still the most used format for data exchange. A big part of this popularity may be attributed to XML Schema language, which is used to describe a structure of an XML document. The ability of XML Schema to formally describe XML documents made XML a core stone of many World Wide Web Consortium (W3C) standards and an essential part of Web interoperability.

The interoperability between different software applications on the Web greatly benefits from Web Services Description Language (WSDL) which provides a formal description of service operations and data formats for defining the message structure. Although WSDL does not impose XML Schema as the only supported type system, usage of other serialization formats is out of scope of the standard, which makes XML Schema the most prevalent format for Web services data type definition [1].

An XML Schema defines an XML document structure providing a set of constraints on its parts and provides a high level of syntactic interoperability. Being a good means for defining data structure, XML Schema lacks semantic expressiveness. To assign semantic meaning to XML Schema elements, Semantic Annotations for WSDL and XML Schema (SAWSDL) is usually used. SAWSDL doesn’t impose any semantic language providing a way to link described elements with their external semantic descriptions. Semantic annotations usually refer to a domain knowledge ontology defined in some ontology language.

OWL 2 Web Ontology Language (OWL 2) is the latest ontology language standard from W3C and one of the most popular ontology languages in biomedicine [2]. Unlike XML Schema, which purpose is to define a structure of a document, ontologies are used to define knowledge domains via logical axioms. The syntactic representation of information in ontology is very vague and may differ not only in serialization format, but also in the way it is defined by the ontology developer. This indeterminacy of syntax impedes OWL 2 direct usage as a WSDL type system and maintains ontology world largely separated from the generation of bioinformatics tools. However, a thoughtful analysis of domain data is, indeed, a requirement to create an appropriate XML Schema. When a domain knowledge is already described in an ontology, this information might greatly facilitate XML Schema development. OWL 2 is designed to use XML Schema datatypes and many of its concepts may be directly translated into XML Schema constructs. Although XML Schema and OWL 2 are very different in their purpose, automatic XML Schema generation based on an OWL 2 ontology definition is feasible.

Ontology based semantic web services development

Conventional Web services provide a high level of syntactic interoperability, but usually are very poor in providing semantic meaning. The idea to describe Web services via ontology languages led to various initiatives [3] consolidated under the term Semantic Web Services.

Nowadays all these pure semantic initiatives, W3C adopted more conservative approach based on extending WSDL descriptions with semantic annotations and providing RDF mapping of WSDL descriptions [4]. However, because Web Services datatypes are still bound to the rigid syntax provided by XML language they cannot be expressed in the WSDL 2.0 OWL ontology.

The requirement of syntactic interoperability doesn’t eliminate the possibility to specify datatypes via ontology languages. OWL 2 provides enough expressiveness to describe data structures, even though most biological ontologies are limited to describe taxonomies and do not contain datatype properties [5]. Properly designed OWL 2 ontology may be transformed into appropriate XML Schema datatype definitions, thus providing the missing piece in Semantic Web Services development.

Usual Semantic Web Services development process consists of various steps such as datatypes schema design, formal Web service description and annotation, etc. The proposed Web Services development workflow (Figure 1) was intended to simplify the development process reducing the need of manual XML Schema design.
Figure 1 Semantic Web Service development

Several conditions should however be met for this workflow to be possible. In the first place, some rules in the building of the ontology itself should be followed, and secondly a procedure to extract a correct and usable XML schema from ontology definitions must be outlined. In this paper, we will discuss recommendations in the building of bioinformatics ontologies to make them usable to derive an ontology-compatible XML Schema for web services, and present the OWL2XS tool to perform an automatic generation of such XML Schema. As an illustrative example, a simple ontology for bioinformatics operations is presented and a complete process of creation of semantically annotated Basic Local Alignment Search Web service (based on the NCBI BLAST tool) is provided (Supplementary Tutorial).

METHODOLOGY

Simple Ontology for Biological Objects

Although OWL2XS tool can be applied to any OWL 2 ontology, the most prospective results may be obtained from the ontologies that incorporate datatypes properties. No matter how thoroughly the ontology is thought-out, Web services require scalar data to be passed as parameters in a form of XML document. Because XML document is essentially a tree structure, the main rule that a bioinformatics operations ontology should meet is to be usable to derive actual Web services is to avoid multiple inheritance. Multiple inheritance is an oversold solution to complex data relationships, and relies in the fact that most ontologies are meant to be understood by humans, who may extract the correct meaning even when it is ambiguously expressed. This is not the case when automatic parsing of the ontology is necessary. A straightforward solution for multiple inheritance is composition. One of the parents should be chosen for the main inheritance, while others are added as additional components or restrictions (Figure 2).

The contents can be still easily interpreted, but also can translate into properly defined data types. This rule must be applied for both classes and properties hierarchies.

Figure 2 Replace inheritance with composition

Figure 3 shows a “Protein sequence record” class in EDAM ontology where multiple inheritance is being used. Here “Protein sequence record” is at the same time a “Protein sequence” and an “Sequence record”. An alternative definition avoiding multiple inheritance would draw a “Protein sequence record” as a “Protein sequence” with property “metadata” of type “Sequence metadata”.

Figure 3 “Protein sequence record” class in EMBRACE Data and Methods Ontology (EDAM).

Although choosing the most adequate layout may not be trivial in some cases, most of the popular data types used in bioinformatics could be defined avoiding multiple inheritance. An example of such style can be found in the BioMoby ontology (Figure 4), that was in fact constructed in the opposite direction, starting from the data types.

Figure 4 BioMoby “AminoacidSequence” class with “Annotations” property.

To illustrate this further we have prepared an example Simple Ontology for Biological Objects (SOBO) (Figure 5). SOBO is greatly influenced by the BioMoby datatypes ontology and follows EMBRACE Data and Methods Ontology (EDAM) [6] architecture, implementing “data”, “format” and “parameter” concepts and provides enough data properties for the appropriate XML Schema generation.

SOBO consists of 38 biological classes, 13 properties and 23 datatypes and may cover most basic bioinformatics operations like sequence manipulation, annotation and comparison, SOBO example ontology can be easily extended with other biological objects definitions and formats using usual tools like the Protégé OWL ontology editor.
Figure 5 SOBO datatypes graph

XML Schema generation

In contrast to ontology languages that describe domain knowledge in a form of logical statements, XML Schemas describe the structure of XML documents that form XML messages conveyed by Internet protocols in Web services. While an XML Schema may be extended, it is generally assumed to be a closed world where something that is not defined should not appear in the XML document. On the other hand, ontology languages are based on an open world assumption where everything that is not implicitly negated is considered possible. Thus, to successfully generate an XML Schema, an ontology must be treated as closed one, assuming that it represents the complete knowledge available.

Considering the nature of ontologies to grow and extend the knowledge, to provide Web services interface stability, it would be reasonable to maintain a well-defined core, which provides datatype descriptions usable for Web services generation.

Although, many OWL 2 entities such as classes, datatypes, properties, and individuals, may be directly translated into XML Schema constructs (Table 1), the translation is nevertheless non-trivial. OWL2XS does the best efforts to produce a coherent XML Schema, notifying about possible translation problems.

Table 1 Overview of the OWL2XS Transformation Model.

<table>
<thead>
<tr>
<th>OWL 2 Entities</th>
<th>XML Schema Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Class</td>
<td>xs:complexType</td>
</tr>
<tr>
<td>owl:Datatype</td>
<td>xs:simpleType</td>
</tr>
<tr>
<td>owl:ObjectProperty</td>
<td>xs:element</td>
</tr>
<tr>
<td>owl:DataProperty</td>
<td>xs:attribute, xs:element</td>
</tr>
</tbody>
</table>

Basic modeling element transformation table.

OWL 2 ontology may comprise of a set of entities that belong to different domains. These entities are usually declared in different namespaces, thus translated into different XML Schemas (Table 2). For instance, SORO ontology models three different concepts and results in three XML Schema documents.

Table 2 OWL 2 to XML Schema translation example.

<table>
<thead>
<tr>
<th>OWL 2 classes</th>
<th>XML Schema</th>
<th>Class: dataStructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>data:ODT</td>
<td>data:ODT</td>
<td>&lt;complexType name=&quot;Structure&quot; summary=&quot;modReference&quot;&gt;data:ODT&lt;/complexType&gt;</td>
</tr>
</tbody>
</table>

Automatically generated XML Schemas may be used for Semantic Web Services development using any convenient toolkit. For the example NCBI Blast Web service, Java API for XML Web Services (JAX-WS) tool has been used.

OWL2XS performs an ontology analysis via ontology reasoner and uses extensive DL (Description Logic) querying in order to determine effective owners and possible ranges of analyzed properties (Table 3). The approach consists in the usage of negation queries to restrict both domains and ranges of the analyzed properties.

Table 3 Example of performed DL queries.

<table>
<thead>
<tr>
<th>Description Logic queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>class: AnnotationSite a class;</td>
</tr>
<tr>
<td>1. 1AminoAcidSequence</td>
</tr>
<tr>
<td>2. 1AminoAcidSequence</td>
</tr>
<tr>
<td>3. 1AminoAcidSequence</td>
</tr>
<tr>
<td>4. 1AminoAcidSequence</td>
</tr>
</tbody>
</table>

Example OWL 2 class representation in XML Schema example.

<table>
<thead>
<tr>
<th>OWL 2</th>
<th>XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;complexType name=&quot;Structure&quot; summary=&quot;modReference&quot;&gt;class: AnnotationSite&lt;/complexType&gt;</td>
</tr>
</tbody>
</table>

Modelling OWL 2 Class entity as an XML Schema complex type.
**OWL 2 Datatypes translation**

OWL 2 Datatypes properties associate an individual with some data value. The type of those values is defined as OWL Datatype. In most cases, the corresponding XML Schema type may be directly used. Custom OWL datatypes may be represented as XML Schema simple type (Table 5). Note that while two equally defined XML Schema simple types are different, this is not the case for the ontology where they are treated as the same datatype.

<table>
<thead>
<tr>
<th>Table 5</th>
<th>OWL 2 Datatype representation in XML Schema</th>
</tr>
</thead>
</table>

**XML Schema**

```xml
<complexType name="AnnotatedAlignment">
  <complexContent>
    <extension base="AnnotatedSequence">
      <sequence>
        <element name="antigenic_annotation" type="AntigenicAnnotation" minOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

*Modeling OWL 2 Datatype as an XML Schema simple type.*

**OWL 2 Class inheritance translation**

As seen before, inheritance is a very powerful modeling concept, and is a key concept both in ontologies and data-type definitions. Although both OWL 2 and XML Schema support inheritance, the latter has some limitations. In OWL 2 language, all derivations are performed via restrictions. This follows from the open world assumption, where a class may contain any property unless it is restricted (i.e., owl:Thing). On the other hand, XML Schema constrains XML document with a closed world assumption. In XML Schema derivations are performed via either an extension or a restriction. Because XML Schema has no multiple inheritance, there is no way to use extensions and restrictions at the same time.

**A. Inheritance via extension**

```
AnnotatedAlignment
```

**B. Inheritance via restriction**

```
SequenceAlignment restriction
```

**C. Complex inheritance**

```
MultipleSequenceAlignment
```

**D. Inheritance breakdown**

```
SubClassOf
```

To get around this limitation, OWL2XS introduces an intermediate type which aggregates the restrictions. OWL2XS employs four different patterns to translate OWL Class inheritance into XML Schema type derivation (Figure 6).

In a simple case, when a new property is introduced into the subtype, OWL 2 Class inheritance translates into XML Schema type extension (Table 6).

| Table 6 | OWL 2 inheritance via XML Schema type extension |

**XML Schema**

```xml
<complexType name="AnnotatedAlignment">
  <complexContent>
    <extension base="AnnotatedSequence">
      <sequence>
        <element name="antigenic_annotation" type="AntigenicAnnotation" minOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

*The "AntigenicAnnotationSequence" class is defined as a sub-class of "AnnotatedAlignment" class that has a mandatory "antigenicAnnotation" property. In XML Schema terms, "AntigenicAnnotationSequence" extends "AnnotatedAlignment" with a new element (see Figure 6 A).*

When the property is restricted, inheritance is translated into XML Schema type restriction (Table 7).

| Table 7 | OWL 2 inheritance via XML Schema type restriction |

**XML Schema**

```xml
<complexType name="SequenceAlignment">
  <complexContent>
    <restriction base="AnnotatedAlignment">
      <sequence>
        <element name="alignment" type="Alignment"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

*The "SequenceAlignment" class extends "Alignment" with a mandatory new element "alignment" (see Figure 6 B).*

Figure 6 Modeling OWL 2 Inheritance in XML Schema.
Along with introducing a new “metadata” property, “SomeOtherAlignment” overrides the “alignment” property found in a parent class “SequenceAlignment”. However, XML Schema cannot model this additional class “SomeOtherAlignment_restriction” as required (see Figure 4 C).

XML Schema supports only single inheritance. An OWL 2 Class that has several parents is mapped into correspondent XML Schema complex type with no parents (Table 9). All the properties from the OWL 2 Class and its antecedents are copied into the XML Schema complex type. This is a standard pattern in XML Schema development, even though it may lead to an inconsistent XML Schema when the generated complex type participates in type subsumption.

Table 9 OWL 2 multiple inheritance in XML Schema

<table>
<thead>
<tr>
<th>OWL 2</th>
<th>XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A SubClassOf Thing (names only xsd:string)</td>
<td>Class A SubClassOf Thing (names only xsd:string)</td>
</tr>
<tr>
<td>Class B SubClassOf Thing (names only xsd:string)</td>
<td>Class B SubClassOf Thing (names only xsd:string)</td>
</tr>
<tr>
<td>Class C SubClassOf A, B</td>
<td>Class C SubClassOf A, B</td>
</tr>
</tbody>
</table>

Unlike the OWL 2 language, XML Schema has no multiple inheritance support. Multiple inheritance may be simulated by collecting all properties from superclasses (“A”, “B”) into the child class “C” (see Figure 4 C).

OWL 2 Properties translation

Properties in OWL 2 represent either relationships between two individuals (Object Properties) or between an individual and some data values (Datatype Properties). In this way properties are aligned into XML Schema elements (Table 10). Cardinality constraints of properties are translated into XML Schema element minOccurs / maxOccurs attributes. In order to minimize
computational complexity/cardinality constraints are limited to the OWL-Lite ones ("0", "1", "unbounded").

Table 10 OWL 2 Properties representation in XML Schema

OWL 2

<complexType name="Sequence">
  <complexContent>
    <restriction base="xs:Data">
      <sequence/>
    </restriction>
  </complexContent>
</complexType>

XML Schema

The "Sequence" type is defined as an extension of the "Data" introducing two local elements "sequence" and "length".

OWL 2 properties may form hierarchies. Properties that participate in a property hierarchy are modeled as global elements with the corresponding substitution groups (Table 11). Although multiple type substitutions are not supported by XML Schema, property inheritance is hardly used in real world ontologies.

Table 11 OWL 2 Properties inheritance as XML Schema element substitutions

OWL 2

<complexType name="Alignment">
  <complexContent>
    <restriction base="xs:Data">
      <sequence/>
    </restriction>
  </complexContent>
</complexType>

XML Schema

The "Sequence" type extends the "Data" type with "sequence" and "length" (see also Table 10).

Usage of local elements and attributes may put restrictions on future ontology extensions. The latter may happen when in a new ontology revision the property becomes a part of property hierarchy which require global element usage and as a consequence renders incompatible XML Schema revisions.

IMPLEMENTATION

The tool is implemented in Java language as Java applet (Figure 7) and uses Apache XML Schema 2.0 (http://ws.apache.org/commons/xmlschemav20) and OWL API 2.1 along with Hermit 1.3.8 reasoner [8] libraries.
Figure 7 OWL2XS XML Schema generator applet.

RESULTS
Automatic XML Schema generation from OWL2 provides web service developers with a power tool for Semantic Web services development. The library itself can be integrated into any java project. A graphical conversion tool that generates XML Schema from a selected ontology file is also provided. OWL 2 EMBRACE Data and Methods Ontology (EDAM) definitions were used as a basement for the example biological ontology provided. This ontology follows EDAM’s architecture, implementing “data”, “format” and “parameter” concepts and provides enough data properties for the appropriate XML Schema generation. Finally, as a usage tutorial, a semantically annotated Basic Local Alignment Search web service (based on the NCBI Blastp tool) is provided (Supplementary Tutorial).

RELATED WORK
The interest in OWL and XML Schema interoperability is not new and a lot of work has been done in representation of XML Schema in OWL-DL [9]. The possibility to transform OWL ontology into XML Schema was proposed [10] as an alternative to XSL Transformations for Semantic Markup for Web Services. A big work to define XML Schema syntax for biological data has been done by BioXSD developers [11], who manually crafted an XML Schema with consequent semantic annotation. A manual XML Schema creation targeting an ontology is a major work that, even in the absence of datatype information in the ontology, may be greatly facilitated by the OWL2XS tool.

DISCUSSION
The automatic XML Schema generation described here provides a consistent and efficient route to a schema, where an ontology that includes datatype information exists. For ontologies that lack datatype information and do not define data models, OWL2XS tool may be used for quick XML Schema prototyping with further manual schema enhancement. Given that biological ontologies continuously incorporate new objects, this approach helps to maintain a consistency between an ontology and the correspondent XML Schema.

A similar functionality may be achieved using SAWSDL Schema Mapping and was mentioned in OWL-S [12]. However, the Extensible Stylesheet Language Transformations (XSLT) proposed for Schema mapping does not suit well for ontological models that can vary in serialization formats. A lack of Web service frameworks support for run-time Schema mapping is also a limitation for this method.

Another approach taken in Semantic Automated Discovery and Integration framework - SADI [13] consists in a direct ontology objects (individuals) service interchange where a service acts as a semantic object consumer/producer. In contrast to OWL-S, SADI does not make any intention to provide WSDL mapping, which restricts its usage by those potential clients who are adherent to conventional Web services architecture.

Unlike other approaches, OWL2XS provides a simple and practical solution for Semantic Web Services development based on ontological datatype-models.

FUTURE DEVELOPMENTS
Future improvements lie in extending OWL 2 support (i.e. OWL-DL cardinality) and better XML Schema support (i.e. XML Schema 1.1 substitutions groups). These improvements may greatly facilitate future Web service development when combined with XML Java Compiler (XJC) extensions to provide a better Java beans generation for generated schema.

FUNDING
This work was supported by the Spanish National Institute of Bioinformatics, Plataforma de Recursos Biomoleculares y Bioinformáticos (PRBB) Instituto de Salud Carlos III - www.inab.org.

REFERENCES
BEING QUA BEING